Sven Casteleyn
Gustavo Rossi
Marco Winckler (Eds.)

LNCS 8541

Web Engineering

14th International Conference, ICWE 2014
Toulouse, France, July 1–4, 2014
Proceedings

Springer

Sven Casteleyn Gustavo Rossi
Marco Winckler (Eds.)

Web Engineering

14th International Conference, ICWE 2014
Toulouse, France, July 1-4, 2014
Proceedings

Springer

Volume Editors

Sven Casteleyn
INIT – Universitat Jaume I
Av. de Vicent Sos Baynat, s/n 12071, Castelló de la Plana, Spain
E-mail: sven.casteleyn@uji.es

Gustavo Rossi
Lifia – Universidad de La Plata
calle 50 y 115, La Plata, Prov. Buenos Aires, Argentina
E-mail: gustavo@lifia.info.unlp.edu.ar

Marco Winckler
ICS-IRIT – Université Paul Sabatier
118 route de Narbonne, 31062 Toulouse Cedex, France
E-mail: winckler@irit.fr

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-319-08244-8 e-ISBN 978-3-319-08245-5
DOI 10.1007/978-3-319-08245-5
Springer Cham Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014941113

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Foreword

The Web plays an important role in every aspect of contemporary societies and of everyday life, i.e., in business, education, entertainment, health, and other critical activities. Web engineering, as a sub-discipline of software engineering, seeks to improve software development for this pervasive, ever-evolving platform, and strives to develop and uncover novel and cost-effective processes, models, methods, and methodologies to support rich, user-friendly, and accessible interactions between people, software, and things.

This volume collects the research articles, late-breaking results, tool demonstrations, posters, tutorials, and keynote speeches presented at the 14th International Conference on Web Engineering (ICWE 2014), held in Toulouse, France, during July 1–4, 2014.

ICWE is the flagship conference for the Web engineering community. Previous editions of ICWE took place at Aalborg, Denmark (2013), Berlin, Germany (2012), Paphos, Cyprus (2011), Vienna, Austria (2010), San Sebastian, Spain (2009), Yorktown Heights, NY, USA (2008), Como, Italy (2007), Palo Alto, CA, USA (2006), Sydney, Australia (2005), Munich, Germany (2004), Oviedo, Spain (2003), Santa Fé, Argentina (2002), and Cáceres, Spain (2001). The 2014 edition of ICWE was centered around the theme of "Engineering the Web for Users, Developers and the Crowd," hereby highlighting the importance of all the different people that, somehow, participate in the development process of interactive Web applications and, ultimately, becomes the actors and the main users of the best practices and results of the research performed in the domain of Web engineering.

ICWE 2014 featured six research tracks, namely: Cross-Media and Mobile Web Applications, HCI and the Web, Modeling and Engineering Web Applications, Quality Aspects of Web Applications, Social Web Applications and Web Applications Composition and Mashups that aimed to focus expertise and create a strong identity for the Web engineering community.

ICWE 2014 was endorsed by the International World Wide Web Conferences Steering Committee (IW3C2), the International Society for Web Engineering (ISWE), the Special Interests Groups on the Web (SIG Web) and Human-Computer Interaction (SIGCHI) of the Association for Computing Machinery (ACM), who provided the in-cooperation agreement to the conference.

ICWE 2014 attracted 100 submissions distributed over the six research tracks. Each paper was assessed by at least three members of an international panel of experts. The Program Committee accepted 20 contributions as full research papers and 13 late-breaking result papers. Additionally, ICWE 2014 welcomed 15 contributions in the form of posters and/or demonstrations, and four contributions to the PhD symposium where young research in the field of Web engineering could benefit from the advice and guidance of experts in the field.

Continuing with a healthy tradition of the ICWE conference series, three tutorials on cutting-edge topics on the field of Web engineering were presented, covering the following topics: Interaction Flow Modeling Language (IFML), Mashups and Web of Things. Moreover, three workshops were selected to be co-located at ICWE 2014.

This high-quality program would not have been possible without the help of many people that assisted the Organizing and Program Committees. We would like to thanks Marc Najork (Google) and Ricardo Baeza-Yates (Yahoo research), our keynote speakers, who accepted to give an inspiring speech at ICWE 2014, of which a written record is included in these proceedings. Many thanks to the Steering Committee liaisons Daniel Schwabe and Marco Brambilla for their advice and moral support to the organization of ICWE 2014 in Toulouse. Our sincere thanks also go out to the local organizer David Navarre, whose support was essential in hosting this conference at the University Toulouse Capitole as well as to Marlène Giamporcaro and Marie-Anne Laplaine, who oversaw all the logistic operations. We also thank Michael Krug and Martin Gaedke for the logistics required for hosting the conference website. Moreover, we address our final thanks to all the authors who submitted their scientific work to ICWE 2014, and especially to the presenters who took the time to come to Toulouse and discuss their work with their peers.

May 2014

Sven Casteleyn
Gustavo Rossi
Marco Winckler

Organization

Technical Committee

General Chair

Marco Winckler	ICS-IRIT, Université Paul Sabatier, France

Program Chairs

Sven Casteleyn	Universitat Jaume I, Castellón, Spain
Gustavo Rossi	UNLP, Argentina

Track Chairs

Cross-Media and Mobile Web Applications

Niels Olof Bouvin	Aarhus University, Denmark
In-young Ko	Korea Advanced Institute of Science and Technology, South Korea

HCI and the Web

Jose Antonio Gallud	Universidad de Castilla La Mancha, Spain
Fabio Paternò	C.N.R.-ISTI, Italy

Modeling and Engineering Web Applications

Marco Brambilla	Politecnico di Milano, Italy
Manuel Wimmer	Vienna University of Technology, Austria

Quality Aspects of Web Applications

Silvia Abrahão	Universidad Politecnica de Valencia, Spain
Filomena Ferrucci	Università di Salerno, Italy

Social Web Applications

Maria Bielikova	Slovak University of Technology in Bratislava, Slovakia
Flavius Frasincar	Erasmus University Rotterdam, The Netherlands

Web Applications Composition and Mashups

Cesare Pautasso	University of Lugano, Switzerland
Takehiro Tokuda	Tokyo Institute of Technology, Japan

Tutorials Chairs

Luis Olsina	Universidad National de la Pampa, Argentina
Oscar Pastor	Universidad Politecnica de Valencia, Spain

Workshops Chair

Santiago Meliá	University of Alicante, Spain

Demos AND Posters

Jordi Cabot	Inria/École des Mines de Nantes, France
Michael Nebeling	ETH, Switzerland

PHD Symposium Chairs

Cinzia Cappiello	Politecnico di Milano, Italy
Martin Gaedke	Technische Universität Chemnitz, Germany

Program Committee

Cross-Media and Mobile Web Applications

Wei Chen	Agricultural Information Institute, Chinese Academy of Agricultural Sciences, China
Antonella De Angeli	University of Manchester, UK
Volker Gruhn	Universität Duisburg-Essen, Germany
Célia Martinie	ICS-IRIT, Université Paul Sabatier, France
George Pallis	University of Cyprus, Cyprus
Fabio Paternò	ISTI-CNR, Pisa, Italy
Benjamin Satzger	Microsoft, USA
Quan Z. Sheng	University of Adelaide, Australia
Beat Signer	Vrije Universiteit Brussel, Belgium
Giovanni Toffetti Carughi	IBM Research Haifa, Israel
William Van Woensel	Dalhousie University, Canada
Marco Winckler	ICS-IRIT, Université Paul Sabatier, France

HCI and the Web

Julio Abascal	University of the Basque Country, Spain
Simone Barbosa	Pontificia Universidade Catolica do Rio de Janeiro, Brazil
Giorgio Brajnik	University of Udine, Italy
Carlos Duarte	University of Lisbon, Portugal
Cristina Gena	University of Turin, Italy
Luis Leiva	Universitat Politècnica de València, Spain
Maria Lozano	University of Castilla-la Mancha, Spain
Maristella Matera	Politecnico di Milano, Italy
Michael Nebeling	ETH Zurich, Switzerland
Victor Penichet	University of Castilla-La Mancha, Spain
Carmen Santoro	CNR-ISTI, Italy
Markel Vigo	University of Manchester, UK
Marco Winckler	ICS-IRIT, Université Paul Sabatier, France

Modeling and Engineering Web Applications

Luciano Baresi	Politecnico di Milano, Italy
Devis Bianchini	University of Brescia, Italy
Hubert Baumeister	Technical University of Denmark, Denmark
Alessandro Bozzon	Politecnico di Milano, Italy
Jordi Cabot	IInria École des Mines de Nantes, Italy
Richard Chbeir	LE2I-CNRS, France
Florian Daniel	University of Trento, Italy
Oscar Diaz	University of the Basque Country, Spain
Schahram Dustdar	Vienna University of Technology, Austria
Jutta Eckstein	IT communication, Germany
Marina Egea	Atos Research & Innovation Department, Spain
Flavius Frasincar	Erasmus University Rotterdam, The Netherlands
Piero Fraternali	Politecnico di Milano, Italy
Irene Garrigós	University of Alicante, Spain
Michael Grossniklaus	University of Konstanz, Germany
Guy-Vincent Jourdan	University of Ottawa, Canada
Gerti Kappel	Vienna University of Technology, Austria
Alexander Knapp	Universität Augsburg, Germany
Frank Leymann	University of Stuttgart, Germany
Maristella Matera	Politecnico di Milano, Italy
Santiago Melia	University of Alicante, Spain
Oscar Pastor	Universidad Politecnica de Valencia, Spain
Vicente Pelechano	Universidad Politecnica de Valencia, Spain
Alfonso Pierantonio	University of L'Aquila, Italy
Werner Retschitzegger	Johannes Kepler University of Linz, Austria

Fernando Sánchez	Universidad de Extremadura, Spain
Daniel Schwabe	PUC Rio, Brazil
Antonio Vallecillo	University of Málaga, Spain
Agustin Yague	Universidad Politecnica de Madrid, Spain
Gefei Zhang	arvato systems, Germany
Jürgen Ziegler	University of Duisburg-Essen, Germany

Quality Aspects of Web Applications

Joao Araujo	Universidade Nova de Lisboa, Portugal
Rami Bahsoon	University of Birmingham, UK
Michela Bertolotto	University College Dublin, Ireland
Davide Bolchini	Indiana University, USA
Giorgio Brajnik	University of Udine, Italy
Cinzia Cappiello	Politecnico di Milano, Italy
Schahram Dustdar	TU Wien, Austria
Carmine Gravino	University of Salerno, Italy
Emilio Insfran	Universitat Politècnica de València (DSIC-UPV), Spain
Tahar Kechadi	University College Dublin, Ireland
Nora Koch	Ludwig Maximilians University of Munich, Germany
Grace Lewis	Carnegie Mellon Software Engineering Institute, USA
Maristella Matera	Politecnico di Milano, Italy
Emilia Mendes	Blekinge Institute of Technology, Sweden
Ali Mesbah	University of British Columbia, Canada
Luis Olsina	GIDIS_Web, Engineering School, UNLPam, Argentina
Federica Sarro	University College, London, UK
Giovanni Toffetti Carughi	University of Lugano, Switzerland
Giuliana Vitiello	University of Salerno, Italy
Michael Weiss	Carleton University, Canada
Coral Calero	Universidad de Castilla-La Mancha, Spain
Arie van Deursen	Delft University of Technology, The Netherlands
Vahid Garousi	University of Calgary, Canada
Jean Vanderdonckt	Université Catholique de Louvain, Belgium
Cristina Cachero	Universidad de Alicante, Spain

Social Web Applications

Witold Abramowicz	Poznan University of Economics, Poland
Ioannis Anagnostopoulos	University of Thessaly, Greece
Marco Brambilla	Politecnico di Milano, Italy

Richard Chbeir	Le2i - CNRS, France
Alexandra Cristea	University of Warwick, UK
Oscar Diaz	University of the Basque Country, Spain
Stefan Dietze	L3S Research Center, Germany
Roberto De Virgilio	Università di Roma Tre, Italy
Vania Dimitrova	University of Leeds, UK
Martin Gaedke	Chemnitz University of Technology, Germany
Geert-Jan Houben	Delft University of Technology, The Netherlands
Zakaria Maamar	Zayed University, UAE
Jose Palazzo Moreira de Oliveira	UFRGS, Brazil
Jan Paralic	Technical University in Kosice, Slovakia
Oscar Pastor	Valencia University of Technology, Spain
Davide Rossi	University of Bologna, Italy
Daniel Schwabe	PUC Rio, Brazil
Markus Strohmaier	University of Koblenz-Landau, Germany
Julita Vassileva	University of Saskatchewan, Canada
Erik Wilde	UC Berkeley, USA
Guandong Xu	University of Technology Sydney, Australia
Jaroslav Zendulka	Brno University of Technology, Czech Republic

Web Applications Composition and Mashups

Saeed Aghaee	University of Lugano, Switzerland
Christoph Bussler	MercedSystems, Inc., USA
Florian Daniel	University of Trento, Italy
Oscar Diaz	University of the Basque Country, Spain
Hao Han	Kanagawa University, Japan
Gregor Hohpe	Google, Inc.
Geert-Jan Houben	Delft University of Technology, The Netherlands
Peep Küngas	University of Tartu, Estonia
Maristella Matera	Politecnico di Milano, Italy
Moira Norrie	ETH Zurich, Switzerland
Tomas Vitvar	Czech Technical University of Prague, Czech Repuclic
Eric Wohlstadter	University of British Columbia, Canada
Christian Zirpins	Karlsruhe Institute of Technology, Germany

Additional Reviewers

Saba Alimadadi
Cristóbal Arellano
Marcos Baez
Michele Bianchi
Hugo Brunelière
Dimoklis Despotakis

Local Organizing Committee

Local Chairs

David Navarre	ICS-IRIT, University of Toulouse Capitole, France
Célia Martinie	ICS-IRIT, Paul Sabatier University, France

Operations

Marlène Giamporcaro	INP-Toulouse, France
Marie-Anne Laplaine	INP-Toulouse, France
Nadine Ortega	University of Toulouse 1, France

ICWE Steering Committee Liaisons

Marco Brambilla	Politecnico di Milano, Italy
Daniel Schwabe	PUC-Rio, Brazil

Acknowledgments

The conference chairs and conference organizers would like to thank our sponsors:

Sponsors

Institute of Research in Informatics of Toulouse (IRIT)
interaction-design.org
University of Toulouse Capitole (Toulouse I)
Paul Sabatier University (Toulouse III)
Institut Nationale Polytechnique de Toulouse (INP)

Scientific Sponsors

ACM In-Cooperation with Special Interests Groups SIGCHI and SIGWEB

International Society for Web Engineering (ISWE)

International World Wide Web Conferences Steering Committee (IW3C2)

Table of Contents

A Platform for Web Augmentation Requirements Specification

Diego Firmenich[1,3], Sergio Firmenich[1,2], José Matías Rivero[1,2], and Leandro Antonelli[1]

[1] LIFIA, Facultad de Informática, Universidad Nacional de La Plata
[2] CONICET, Argentina
`{sergio.firmenich,mrivero,lanto}@lifia.info.unlp.edu.ar`
[3] Facultad de Ingeniería, Universidad Nacional de la Patagonia San Juan Bosco
`dfirmenich@tw.unp.edu.ar`

Abstract. Web augmentation has emerged as a technique for customizing Web applications beyond the personalization mechanisms natively included in them. This technique usually manipulates existing Web sites on the client-side via scripts (commonly referred as *userscripts*) that can change its presentation and behavior. Large communities have surfaced around this technique and two main roles have been established. On the one hand there are *userscripters*, users with programming skills who create new scripts and share them with the community. On the other hand, there are *users* who download and install in their own Web Browsers some of those scripts that satisfy their customization requirements, adding features that the applications do not support out-of-the-box. It means that Web augmentation requirements are not formally specified and they are decided according to particular *userscripters* needs. In this paper we propose CrowdMock, a platform for managing requirements and scripts. The platform allows *users* to perform two activities: (i) specify their own scripts requirements by augmenting Web sites with high-fidelity mockups and (ii) upload these *requirements* into an online repository. Then, the platform allows the whole community (*users* and *userscripters)* to collaborate improving the definition of the augmentation requirements and building a concrete script that implements them. Two main tools have been developed and evaluated in this context. A client-side plugin called MockPlug used for augmenting Web sites with UI prototype widgets and UserRequirements, a repository enabling sharing and managing the requirements.

1 Introduction

Nowadays, the Web is a really complex platform used for a great variety of goals. This advanced use of the Web is possible because of the evolution of its supporting technology and the continuous and massive demands from users that enforces and accelerates its rapid evolution. The Web has not evolved only in terms of technical possibilities but also in terms of supporting an increasing number and types of users too. The same crowd of users has evolved in parallel with the Web itself and the user expectations in terms of what the applications provide to them are very demanding nowadays. Even more, users have found ways to satisfy their own requirements when they are not taken into account by Web applications developers. It is not casual that several Web augmentation [5] tools have gone emerging in this context. These tools pursue the goal of modifying existing Web sites with specific content or functionality that were not originally supported. The crowd of users continuously builds a lot of such as tools (distributed within the community as browsers plugins, scripts, etc.). It is clear that users are interested in using the Web in the way they prefer and they have evolved to know how to achieve it. Even in the academy there is a dizzying trend in the area of end-user programming [11]. The reason is that (1) part of the crowd of users may deal with different kinds of development tools, and (2) part of the crowd of users want to utilize these tools in order to satisfy a particular need/requirement.

Although several tools such as Platypus[1] allow users to perform the adaptation of Web pages directly by following the idea of "what you see is what you get", these kinds of tools have some limitations regarding what can be expressed and done. Thus, some Web augmentation requirements could not be addressed with them, and these have to be tackled with other kind of script, which require of advance programming skills to be developed. In this sense, we think that prototyping could be also a solution in the context of Web augmentation requirements, and augmenting Web pages with the prototypes is really a straightway for telling others what a user wants.

In addition, users are more familiarized in dealing with software products, which is partially proven by new agile development methodologies like Scrum [19] that center the development in tackling user needs by prioritizing requirements considering its business value. The main purpose of such processes is providing valuable functionality as early as possible giving a more active role to the stakeholders to assess that the implemented application is what they need and expect. It is usual to ask users for defining a requirement by using one of several techniques such as mockups [20][6] or User Stories [4], which are completely understandable by users. Since agile methodologies try to include stakeholders (including users) as much as possible in the development process, their processes usually are compatible with User-Centered Design (UCD) approaches. Among the most common requirements artifacts used in UCD, low-fidelity prototypes (informally known as mockups) are the most used [9]. Mockups are more suitable than textual requirements artifacts because they are a shared language between development team (including developers and analysts) and users [14], avoiding domain-related jargon that may lead to faults in delivered software artifacts originated by requirements misunderstandings. In addition, mockups have been proposed as a successful tool to capture and register fluid requirements [17] – those that are usually expressed orally or informally and are an implicit (and usually lost) part of the elicitation process. However, they are rarely used in an isolated way, being usually combined with User Stories [4] or Use Cases [13][8]. Finally, UI Mockups have the advantage of being simple since stakeholders can build these by themselves using popular and emergent tools like Balsamiq[2] or Pencil[3].

[1] `https://addons.mozilla.org/es/firefox/addon/platypus/`
[2] `http://balsamiq.com`
[3] `http://pencil.evolus.vn/`

Most of the development approaches regarding using mockup are centered on applications in which the number and type of end-users are predictable. However, this is not true for massive Web applications, since the crowds that use them are not only less predictable in quantity and type, but also are changing and evolving all the time. In order to satisfy as many users as possible, the application should be customized for each user. Although the mechanism of adaptation and personalization of Web applications, the so-called *adaptive Web* [2], have also evolved (for instance, complex user models [7] have been proposed and assessed), it is really hard to give adaptation mechanisms that contemplate the requirements of all the users.

In this context, two trends with two main goals have emerged. On the one hand, well-known mash-ups [23] approaches have surfaced for integrating existing Web content and services. On the other hand, a technique called Web augmentation [5] has emerged in order to allow users to customize both content and functionalities of Web pages modifying their DOMs on the client-side. Both approaches have communities that serve as a proof of how important is for the users to improve their Web experiences. An example of the former is Yahoo Pipes [22], a mash-up tool broadly utilized for combing services over the Web that has a big active community [3]. Regarding Web augmentation, the userscripts.org [21] community represents an emblematic example. This community has created thousand of scripts that run over the client for modifying the visited Web sites. Some scripts have been installed for end users over one hundred thousand times.

In both communities we observed two different user groups: (1) the crowd of users (at least part of them) want to personalize the Web applications they use and, (2) the part of this crowd can develop software artifacts for such purpose. Our main concern is to facilitate how these communities currently work. In the design of Web applications from scratch, different stakeholders participate on the definition of the functionality. This can be done since part of these reduced and well-known set of stakeholders directly *express* their requirements. There is an explicit delegation. On the contrary, most of the open communities that intend to introduce custom enhancements to existing Web sites work usually in the opposite way: a user from the crowd with programming skills develops an artifact with some features that may be useful to other users. If the artifact is good enough and responds to a common requirement, then, it will possibly be installed for many other users of the crowd.

In some cases both user and userscripters work together by asking new requirements and implementing them (correspondently) in two ways: (1) asking for new scripts in the community forums or (2) asking scripts improvements in the script's discussion Web page. In this work we aim to create new communication mechanisms for improve both the process and how resultant scripts match user requirements. The main motivation behind the approach presented here relies in the fact that, besides informal forums, the community has no clear ways to communicate to those users with development skills in the crowd, which are the *augmentation requirements* desired.

In this paper we propose to rely in mechanisms used on agile methodologies, such as User Stories or mockups, to empower users with tools for specifying their enhancements requirements over existing Web applications. The novelty of the approach is the usage of Web augmentation techniques: with our approach, a user can augment a Web page with those widgets that are relevant for describing his requirement. Then, our approach allows sharing the augmentation requirement so other *users* and *userscripters* may reproduce the same augmentation in their own Web browsers in order to understand that user's needs in a more detailed and formal way.

The paper is organized as follows. Section 2 introduces our approach. Section 3 presents the tools, and section 4 shows the results of an evaluation made with end-users. The state of the art is tackled in section 5. Finally, section 6 gives some concluding remarks and further works.

2 The Approach

We present a novel approach for UI prototyping based on Web augmentation. In this work, Web augmentation is used as the technique for defining requirements while Web augmentation communities are taken as the source of both requirements and scripts for these requirements.

Our goal is to improve the communication between *users* and *userscripters*. The essence of the approach can be described through the following example: let's consider that a user is navigating a common video streaming Web site like YouTube. When he searches for videos, he realizes that additional information in the results screen will be helpful for him in order to decide which video to watch. For instance, he would like to see the amount of likes/dislikes and also further information about video's audio quality, resolution, etc. Consider that the user knows how to install and run GreaseMonkey scripts to augment Web sites, but he does not how to develop one of such scripts. If there were no scripts satisfying his concrete expectations, he would like to have a way to communicate to GreaseMonkey scripters his requirements. This communication should be as concrete and clear as possible. Currently, userscripting communities like userscripts.org rely on textual messages interchanges as the solely artifacts to let users and userscripters collaborate. The presented approach proposes to help the end-users and userscripters collaboration through:

- Linking users requirements with *userscripters* who can implement the requirements.
- Using a concrete language for defining those requirements. The language must be understood by both developers and users and must be formal enough to describe the requirement in terms of UI components and features.
- Managing and evolving the requirements in a well-defined process.

Our approach follows the general schema shown in Figure 1. The main goal is that any user can specify the visual components related to a particular requirement by augmenting the target Web page with mockups.

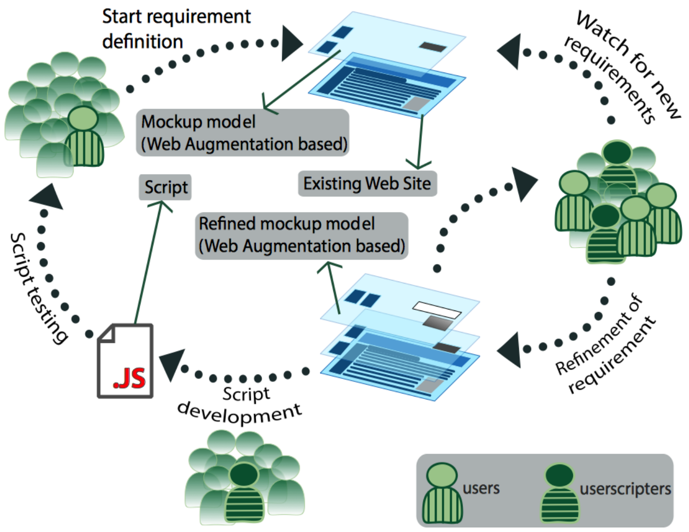

Start requirement definition
Mockup model (Web Augmentation based)
Existing Web Site
Watch for new requirements
Script
Refined mockup model (Web Augmentation based)
Refinement of requirement
Script testing
.JS
Script development
users
userscripters

Fig. 1. Overview of the approach

Once the requirement is concretely defined (which is expressed through a model that will be introduced later); it can be uploaded into an online repository, which makes it accessible by the community to *reproduce* and enhance this. In this context, to reproduce means that any other user can augment the target Web page in the same way the user that specified the requirement did, in order to watch in detail which the required UI alterations are.

The community (i.e. users and userscripters) may colaborate in the refinement of a requirement in the repository. If a user is not capable to define the requirement with the required level of detail, another user who wants to colaborate (for example, having the same requirement) can enrich the definition. Every change is validated by the requirement's owner. When a stable version of the requirements is met, a userscripter may create the script that implements it, and then upload it into the repository. At this point, users can install and use the script, contributing with its testing.

Therefore, our approach empowers users with:

- Mechanisms for defining their requirements over existing Web pages.
- Mechanisms for managing and publishing their requirements.
- Mechanisms for proposing and bringing solutions to users, who can evaluate these solutions and eventually start the cycle again.

The main purpose of our approach, called *CrowdMock*, relies on allowing the *crowd of users* to *plug* high-fidelity *mockups* into existing Web pages. Thus, the approach involves two main contributions. On the one hand, we designed a metamodel (the *MockPlug metamodel*) for describing how high-fidelity mockups are woven into existing Web sites via Web augmentation. On the other hand, to support our approach technologically we developed a client-side tool for weaving augmentation models (called *MockPlug*) and a server-side platform for managing them collaboratively (called *UserRequirements*). MockPlug is a tool designed for allowing end-users to *plug* mockups on existing Web pages to specify their augmentation requirements and them in a concrete, formal model. With MockPlug, users may create, update, and reproduce requirements. UserRequirements is a platform deployed as a Web application[4] and also integrated with MockPlug, for allowing end-users to share their augmentation requirements and asking for scripts that implement them. This also contemplates the evolution of the requirement and the evaluation of a particular solution allowing collaboration, traceability, and validation.

Web augmentation operates over existing Web sites, usually by manipulating their DOM[5]. If we want to augment a DOM with new widgets, we have to specify how those new widgets are woven into the existing DOM. With this in mind, we defined the MockPlug metamodel, which specifies, which kinds of augmentation (widgets and their properties) are possible within our approach and also how these are materialized for a particular Web page's DOM. Within MockPlug metamodel, which is depicted in Figure 2, both the name and the URL of the augmented Web site compose a requirement. The specification of new widgets relevant for the requirement (Widgets) may be defined in the model. Widgets can be simple (SimpleWidgets, atomic and self-represented) or composite (CompositeWidgets, acting as a container of another set of Widgets), but all of them have several characteristics in common:

- Every Widget belonging to a MockPlug model has both a type and properties.
- A Widget is prepared to respond to several events, such as *mouseover* or *click.*
- A Widget can react to an event with predefined operations.

A Widget has information related to how and where it was inserted into the DOM tree. Note that, each widget has an insertion strategy associated (*float*, *leftElement*, *rightElement*, *afterElement*, *beforeElement, etc.*).

It is worth mentioning the importance of the property *url* in a MockPlug model, since this is the property that defines which Web site (o set of Web sites when using a regular expression) the model is augmenting.

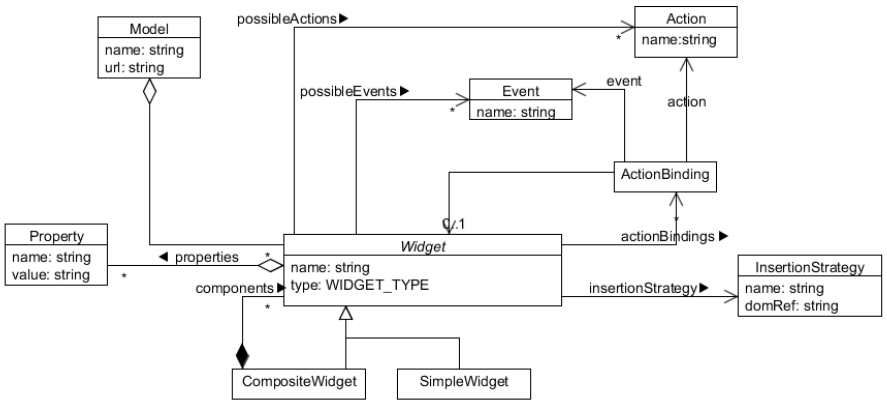


Fig. 2. MockPlug metamodel

[4] See a live demo of our approach on: `http://www.userrequirements.org`

[5] `http://www.w3.org/DOM/`

3 Tools for Web Augmentation Requirements Management

In this section we introduce the technical details behind the aforementioned MockPlug and UserRequirements implementation.

3.1 MockPlug

We considered two issues in order to empower users with mechanisms for defining their own requirements, (1) the supporting tool for such purpose has to be easy to use and (2) specifying a new requirement should not require too much effort and high technical knowledge. We developed MockPlug with this in mind. MockPlug is a Firefox extension that allows end-users to specify requirements by manipulating high-fidelity mockups over their Web sites that they want to augment. For each MockPlug requirement, a User Story is also defined.

From MockPlug's point of view, a requirement represents a set of modifications made over an existing Web site expressed in terms of the aforementioned metamodel. Thus, every requirement expressed in our approach has a MockPlug model associated, and the changes are expressed by adding widgets. From the user's point of view, these have a visual style like hand-drawn mockups similar to mockup tools like Pencil or Balsamiq. However, they are represented at runtime as ordinary DOM elements. In order to refine his requirements visually, a user may drag&drop widgets over the existing Web page, like it is shown in Figure 3. This figure depicts how the user can add a new button over the existing IMDB Web site. On the top of Figure 4, the button already added over the existing Web page is shown and the widget editor is shown at the bottom.

3.1.1 Widgets

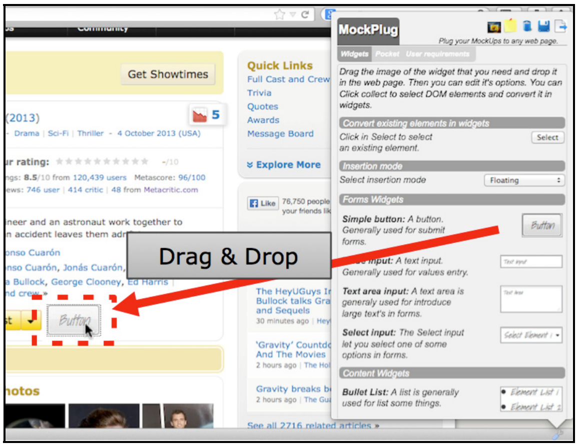
The tool considers three kinds of widgets: *predefined* widgets, *packetized* widgets and *collected* widgets, which are described below.

Predefined Widgets

We have defined an initial set of predefined widgets (Figure 5), which are grouped in categories as Table 1 describes. Different properties may be set for each kind of predefined widget. For instance, the widget *Button* has the properties *name*, *title* and *label;* while a menu list has the *name* and a *collection of options*. Figure 5 shows how the predefined widgets are listed in the MockPlug Panel.

Pocketized Widgets

In order to allow users to easily specify integration requirements we created a functionality called *Pocket*. The Pocket (depicted in Figure 6) allows users to collect existing UI components as widgets. Its name comes from its provided functionality, which consists in allowing collecting and storing any part of an existing Web page and then place it as a *new* widget in other Web applications or pages. As an example, Figure 6 shows three already collected widgets (one collected from YouTube and two collected from IMDB) that could be added in any other Web application in order to define some requirement of integration between these applications. The user must drag&drop the widget that had collected in the same way it did with predefined widgets, having the same tool aid and insertion strategies to apply. Pocketized widgets can be added as static *screenshots* of the original DOM or as real DOM elements. However, so far the tool does not guarantee the correct behavior defined with JavaScript routines taken from the original Web page from where the packetized widget was collected.



Plug your MockUps to any web page.

Drag the image of the widget that you need and drop it in the web page. Then you can edit it's options. You can Click collect to select DOM elements and convert it in widgets.

Convert existing elements in widgets

Click in Select to select an existing element. Select

Insertion mode

Select insertion mode: Floating

Forms Widgets

Simple button: A button. Generally used for submit forms.

Text input: A text input. Generally used for values entry.

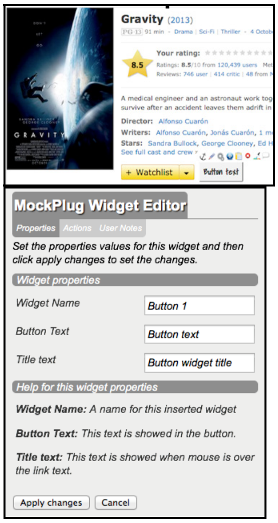
Text area input: A text area is generaly used for introduce large text's in forms.

Select input: The Select input let you select one of some options in forms.

Content Widgets

Bullet List: A list is generally used for list some things.

Fig. 3. MockPlug main panel



Set the properties values for this widget and then click apply changes to set the changes.

Widget properties

Widget Name	Button 1
Button Text	Button text
Title text	Button widget title

Help for this widget properties

Widget Name: A name for this inserted widget

Button Text: This text is showed in the button.

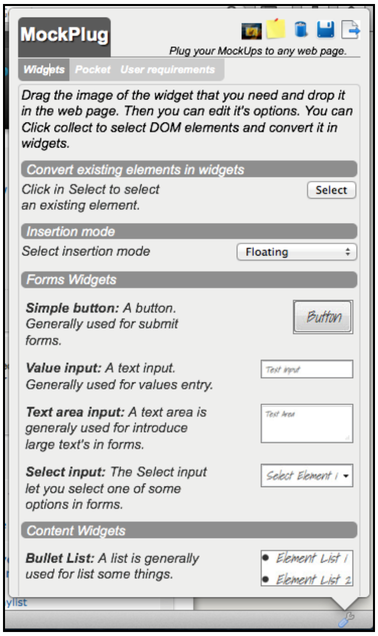
Title text: This text is showed when mouse is over the link text.

Fig. 4. Widget added and widget editor

Table 1. Predefined widgets

Category	Goal	Examples
Form Widgets	Specify requirements that involve some kind of information gathering.	Button, text input, text area, select menu
Content Widgets	Specify requirements focused on non-editable contents	Lists, images, text, anchors
Annotation Widgets	Empower users with mechanisms for describing textually, some expected behavior, presentation or content	Post-its notes, bubble comments

The user may add both predefined and pocket widgets in the DOM by using an insertion strategy. This function provides an interactive highlighting of existing elements in the DOM in order to help the user.


MockPlug
Plug your MockUps to any web page.
Widgets | Pocket | User requirements

Drag the image of the widget that you need and drop it in the web page. Then you can edit it's options. You can Click collect to select DOM elements and convert it in widgets.

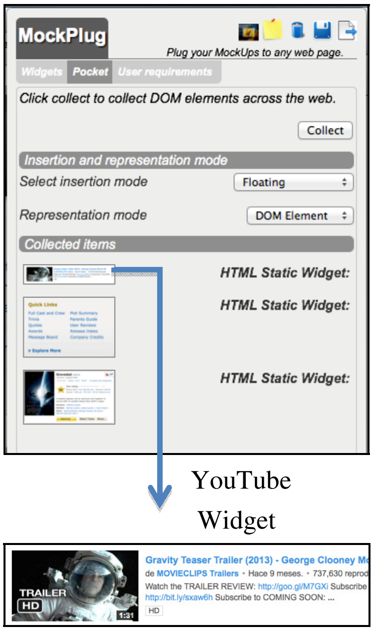
Convert existing elements in widgets
Click in Select to select an existing element. [Select]

Insertion mode
Select insertion mode: Floating

Forms Widgets
Simple button: A button. Generally used for submit forms. [Button]
Value input: A text input. Generally used for values entry. [Text input]
Text area input: A text area is generaly used for introduce large text's in forms. [Text Area]
Select input: The Select input let you select one of some options in forms. [Select Element 1]

Content Widgets
Bullet List: A list is generally used for list some things.
- Element List 1
- Element List 2

Fig. 5. Predefined Widgets panel


MockPlug
Plug your MockUps to any web page.
Widgets | Pocket | User requirements

Click collect to collect DOM elements across the web. [Collect]

Insertion and representation mode
Select insertion mode: Floating
Representation mode: DOM Element

Collected items
HTML Static Widget:
HTML Static Widget:
HTML Static Widget:

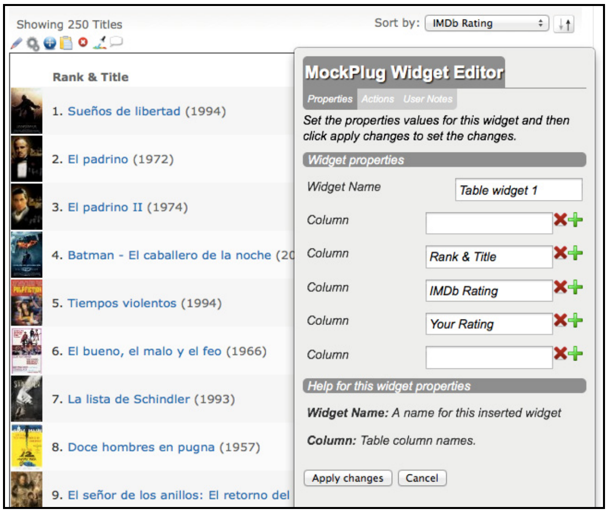
Gravity Teaser Trailer (2013) - George Clooney Mo
de MOVIECLIPS Trailers • Hace 9 meses. • 737,630 reprod
Watch the TRAILER REVIEW: http://goo.gl/M7GXi Subscribe
http://bit.ly/sxaw6h Subscribe to COMING SOON: ...
HD

Fig. 6. Pocket panel

Collected Widgets
Although the Pocket allows users to specify some requirements of integration and also to reuse existing UI components, sometimes it could be useful to convert an existing DOM element into widgets in order to refine it. This can be accomplished using the *collected widgets* functionality provided by the tool. A *collected widget* is a widget that has been created or imported from an existing DOM element in order to manipulate it. For example, let's assume that the user wishes to indicate that a new column should be displayed in an existing table. Then, the user can convert the existing table into a new widget with the final goal of editing its contents and columns, as it is shown in Figure 7. In this case, the user collected the table containing the 250 top movies from IMDB.com.

The ability of converting existing DOM elements into widgets, also gives the possibility of removing useless items from the target web site. Since the DOM element is converted into a widget, it could be also moved to another part of the Web page. Moreover, converting previously existing DOM elements into widgets makes possible to use them as a target or reference in actions define for other widgets. For example, the user might want to add a button to show/hide a particular element of the website on which the requirement is made.

It is worth noting that collecting a DOM element with the goal of creating a collected widget is not the same that putting a widget into the Pocket. While the first one associates the underlying DOM element as a widget in the MockPlug requirements model, the second ones can be used as a sort of *clipboard* for moving UI pieces among Web applications.


Showing 250 Titles

Sort by: IMDb Rating

Rank & Title

1. Sueños de libertad (1994)
2. El padrino (1972)
3. El padrino II (1974)
4. Batman - El caballero de la noche (20
5. Tiempos violentos (1994)
6. El bueno, el malo y el feo (1966)
7. La lista de Schindler (1993)
8. Doce hombres en pugna (1957)
9. El señor de los anillos: El retorno del

MockPlug Widget Editor

Properties | Actions | User Notes

Set the properties values for this widget and then click apply changes to set the changes.

Widget properties

Widget Name	Table widget 1
Column	
Column	Rank & Title
Column	IMDb Rating
Column	Your Rating
Column	

Help for this widget properties

Widget Name: *A name for this inserted widget*

Column: *Table column names.*

Fig. 7. Edition of collected Widget

3.1.2 Widgets Management

The widgets added with MockPlug are clearly distinguishable from the rest of the elements found in the website. However, MockPlug makes it possible to copy styles from existing DOM elements to a Widget in order to emulate an existing page style. In the example from Figure 8, the user has added a button (predefined widget) called “Button text”, which is actually a DOM element (as any widget) and which with the user can interact. Different operations commonly found in mockup tools can be applied through the widget contextual toolbar (shown when the widget is selected), such as cloning, removing, moving, annotating, property editing, among others.

3.1.3 Code Generation

User requirements are abstracted with the MockPlug metamodel and their instances may be processed in order to generate code through MockPlug code generators as in well-known Model-Driven approaches [10]. The code generation capabilities in MockPlug make it possible to generate a first code stub implementing basic and repetitive features of the augmentation requirement, thus reducing the workload for the *userscripter* who wants to write a script to satisfy it. The code generator has been included by default with MockPlug and currently it is focused on obtaining GreaseMonkey scripts.

3.2 Requirements Repository

In addition to MockPlug, we have implemented UserRequirements, a repository with social features with two main purposes: (1) allowing users to collaborate in the definition and the evolution of augmentation requirements and (2) enabling *userscripters* to describe and reproduce the requirements in order to develop the corresponding script. A requirement in our repository is defined by two components in our repository:

- **A User Story (US)**, which highlights the core concerns of the requirement from a particular user role pursuing some goal in the business [4].
- **A MockPlug model**, which is the UI prototype for answering to the US. A MockPlug model is associated with only one US, but one US can be referenced in multiple MockPlug models.

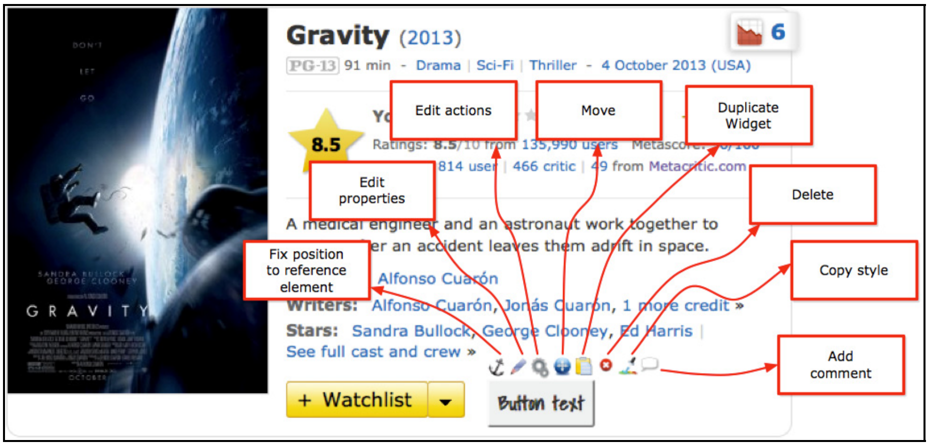
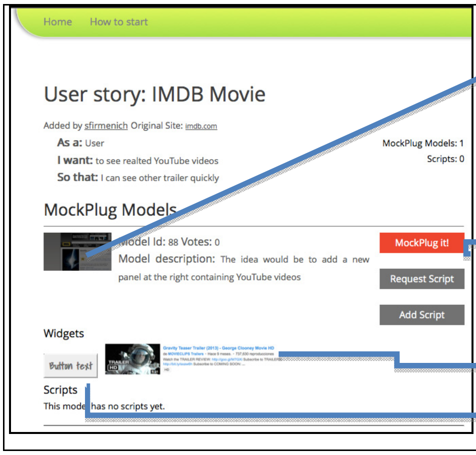


Fig. 8. Widget contextual menu



Home How to start

User story: IMDB Movie

Added by sfirmenich Original Site: imdb.com

As a: User

I want: to see realted YouTube videos

So that: I can see other trailer quickly

MockPlug Models: 1

Scripts: 0

MockPlug Models

Model Id: 88 Votes: 0

Model description: The idea would be to add a new panel at the right containing YouTube videos

MockPlug it!

Request Script

Add Script

Widgets

Button text

Scripts

This model has no scripts yet.

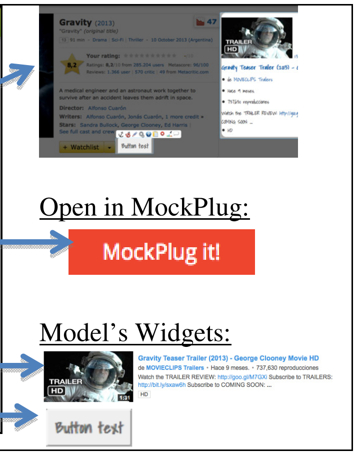
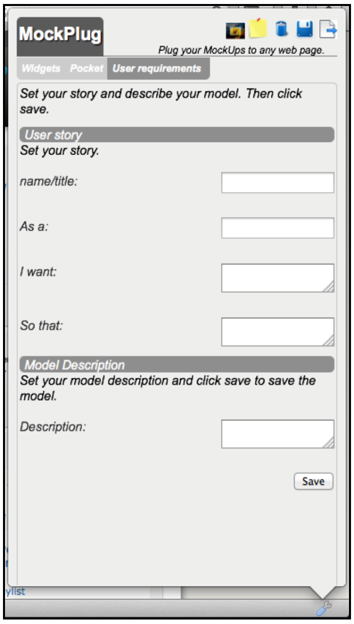


Fig. 9. Requirement view in UR **Fig. 10.** UI details

Figure 9 and 10 shows a screenshot of how a requirement is described in the repository. It includes information about who created the requirement, over which Web site it was defined, its US, and a screenshot of how the target Web site was augmented with new widgets by using MockPlug.

It is really important to allow developers to reproduce and watch the resulting *augmented mockup* in action. This functionality is provided by the button labeled *MockPlug it* (see Figure 10). When the user clicks this button, a new tab is opened in order to load the target Web site and augment it with the MockPlug model built interactively using the MockPlug tool. Others users can collaborate by evolving the definition of that requirement using the tool.

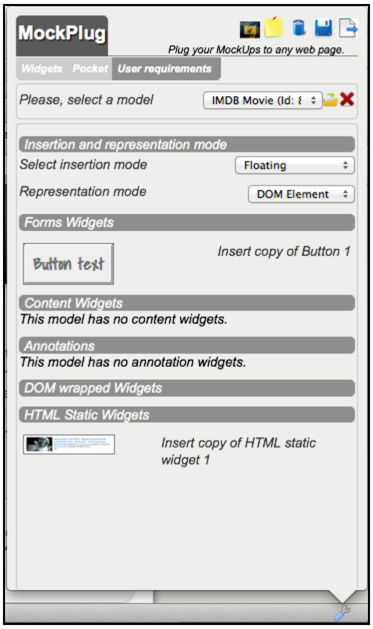
The integration between the social repository and MockPlug tool is essential for our approach. We show how the user can add his current requirement in the repository on the left of Figure 11. To add a new requirement, the user has to define the User Story plus a general description of it. Finally, the requirement (formed by the US and the MockPlug model) is uploaded to the repository by clicking the *Save* button.



Set your story and describe your model. Then click save.

Set your story.

Set your model description and click save to save the model.



Please, select a model: IMDB Movie (Id: 8

Select insertion mode: Floating

Representation mode: DOM Element

This model has no content widgets.

This model has no annotation widgets.

Fig. 11. Integration between MockPlug and UserRequirements

We depict how a user may choose from the menu between his already existing models in the repository on the right in the same figure. When a model is selected, all the widgets involved in that model are listed at the bottom from the same panel. The user also may choose to *open* the MockPlug model, whose action will render it in a new tab. The difference between *pre-open* (just for listing its widgets) and open a model (which actually renders it) was meant on purpose, because by pre-opening a model the user can reuse widgets defined on the model that is being defined.

For conciseness reasons we can not explain it deeply, but it is important to know that UserRequirements make it possible to "branch" and "vote" the requirements models. In this way the community may reach a personalization level of the requirement that fulfills the expectations of the majority while someone other person of the community may create a new branch of one model in order to specify further requirements.

4 Evaluation

This section describes a preliminary evaluation of the CrowdMock approach and its supporting tools. We have performed a controlled experiment focused on assessing the effectiveness achieved by experienced users when specifying requirements on existing Web applications. The experiment compared the process of specifying requirements using the proposed approach against using existing approaches. Our hypothesis was:

> *Defining augmentation requirements with CrowdMock produces better specifications than using traditional user-oriented methods (such as user stories, mockups, etc.), and consequently, CrowdMock improves the understandability of the requirements.*

4.1 Protocol and Participants

The experiment was organized into two phases: (1) defining a requirement using well-known techniques (e.g. user stories, mockups, etc...) and (2) defining requirements using MockPlug. The requirements had to be defined over well-known existing Web applications: IMDB[6] and YouTube[7]; and they had to be based on one of the following features. The features for IMDB were:

- R1.1: Filter the list of 250 best movies
- R1.2: Add some information to the 250 best movies list.
- R1.3: Change the layout and/or content of a movie's page based on the following issues:
    - Move elements of the page.
    - Remove elements from the page
    - Add new widgets into the page (button, input box, menu, etc.)
    - Add contents related to the movie from another IMDB page.

The features for YouTube:

- R2.1: Add information in the search results about the videos.
- R2.2: Idem to R1.3 focusing on a YouTube video's Web page.

Participants had to choose 2 features in each phase. Thus, each participant specified 4 requirements. Since 8 subjects participated in the experiment, a total of 32 requirements were specified. Half of them (16 requirements) had to be defined with the participants' preferred requirements artifacts and elicitation techniques, and the other 16 requirements had to be defined using MockPlug. All the 32 requirements were implemented by means of GreaseMonkey scripts, which were developed by a specific team at LIFIA.

[6] Internet Movie Data Base - `http://imdb.com;` last accessed 4-Feb-2014.
[7] YouTube – `http://youtube.com`; last accessed 4-Feb-2014.

The whole experiment was organized as follows:

1. First phase:
   a. A first face-to-face meeting with participants was organized for presenting the experiment and explaining the tasks. Before this meeting, participants had been introduced in the use of US as well as in user interface mockup building techniques and tools. In this meeting the definition of two requirements choosing one specific technique and one or more requirements artifacts was exemplified.
   b. Participants had 3 days to write two requirements. Then, they had to send their specifications (including the requirements artifacts) and an activity diary where they registered all their work and the time spent on it.
   c. When all the requirements were collected, developers had 2 days to create the 16 GreaseMonkey scripts. Both 2 scripts implementing the requirements were sent to every participant.
   d. Participants had 2 days for installing, using and evaluating both scripts and filling in two questionnaires:
      i. Questionnaire A, oriented to measure and describe the difficulty perceived during the specification of the requirements.
      ii. Questionnaire B: oriented to measure and describe how well the script satisfied his expectations.
2. Second phase:
   a. Another face-to-face meeting was organized for presenting both the new tasks to be accomplished and the tools involved in this phase:
      i. The UserRequirements (UR) application (i.e., the requirements repository) was introduced. Participants were asked to create a user account on the platform in order to start specifying requirements through it.
      ii. MockPlug was also explained and participants learned about how to upload the requirement from MockPlug to UR.
      iii. The second evaluation task was presented. It consisted in defining two requirements using MockPlug and UR. Participants had to choose only requirements not chosen in the first phase.
   b. Participants had 3 days to define the requirements and then upload them to the repository. Then, they had to send separately a document including the activity diary.
   c. With every requirement uploaded to UR by participants, the developers received a notification. Again, developers had 2 days to create the 16 GreaseMonkey scripts. When all the requirements were uploaded, instead of sending the scripts personally, developers uploaded the script for each requirement into UR and participants received the corresponding notification.
   d. Participants had 2 days for downloading and using both scripts. After that, they were asked to filling in the same two questionnaires (A and B). Also, they were asked to fill in another SUS [1] questionnaire for evaluating the usability of our tools.

Following a within subjects design, a total of 8 participants were involved in this evaluation. All participants were professionals on computer science from a post-graduate course on Web Engineering; 4 female and 4 male and aged between 24 and 60. It is important to mention that the 8 participants were instructed (before the experiment) in the installation and use of GreaseMonkey scripts.

4.2 Results

This section describes the results of the experiment. First, we describe the requirements specified by every participant in Table 2. Afterwards, we present the analysis.

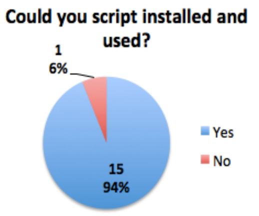
4.2.1 First Phase

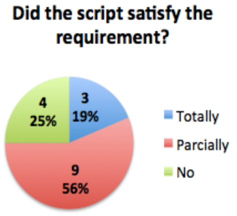
Table 2 shows depicted with "X" the requirements selected in the first phase for each participant. The average time for defining each requirement was 92.5 minutes (SD = 62.12 minutes). The techniques used were mockups (traditional ones), User Stories and some more textual description.

The difficulty perceived when specifying each requirement ranged from "normal" to "very easy". Most of the requirement specifications (11) were considered as a "normal" task in terms of difficulty (68.75%) in a scale from 1 to 5 (where 1="very easy", 2="easy", 3="normal", 4="difficult", 5="very difficult"). This task was considered "easy" for 4 requirements (25%), and "very easy" only for 1 (6.25%).

Table 2. Requirements selected by each participant (X = First Phase, O = Second phase)

	R1.1	R1.2	R1.3	R2.1	R2.2
Participant 1		X	X	O	O
Participant 2		O	O	X	X
Participant 3		O	O	X	X
Participant 4		X	X	O	O
Participant 5		X	O	O	X
Participant 6		X	O	O	X
Participant 7	O	X		X	O
Participant 8		X	O	X	O





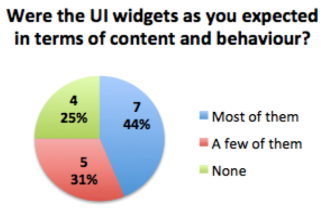


Fig. 12. Phase 1 results

After receiving all the augmentation requirements specifications, we implemented them and sent the scripts to every participant. They had to download, install and use these scripts in order to evaluate them. The results are shown in Figure 12.

4.2.2 Second Phase Results

Requirements chosen for the second phase of the experiment are depicted with "O" in Table 2. As commented previously, participants had to use MockPlug and UserRequirements in this phase. Defining a new requirement took in average 86.56 minutes (SD = 66.35). However, the difficulty perceived in the specification of each requirement had a wide range. The difficulty of the specification task was considerer "normal" (25%) in 4 requirements. Another 25% were considered "difficult" while "very difficult" was used in one requirement (6.25%). Other 37.5% (6 requirements) were qualified as "easy". Finally, 1 requirement specification was considered "very easy".

We believe the reasons were: (1) people were not experienced in the use of Web augmentation tools and (2) some incompatibilities between participants' Firefox extension version and our plug-in. This was finally confirmed by the SUS questionnaire. MockPlug scored 74.9, which is an improvable result regarding the usability of the tool.

Despite of these contingencies, we observed that participants were more satisfied regarding the implementation of their requirements when using MockPlug. It would indicate that the scripts in this phase were closer to user expectations than those developed for the first one. This information is depicted in Figure 13.

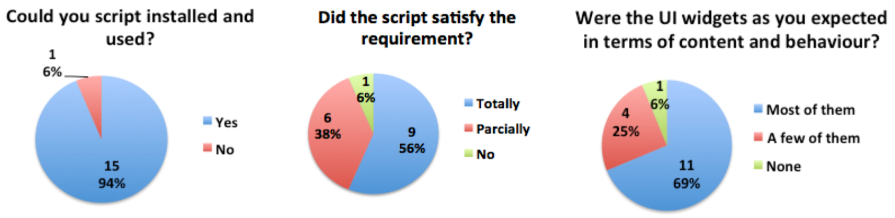


Fig. 13. Phase 2 (CrowdMock) results

4.2.3 Discussion

The results of the controlled experiment showed that when users freely *choose* the specification technique only 18.75% of the requirements were totally satisfied and 56.25% were partially specified. The remaining 25% of the scripts did not satisfy absolutely user requirements.

In the second phase, by contrast, 56.25% scripts fulfilled user's expectations, while 37.5% satisfied them partially. Because one of the scripts was not able to be executed, just one requirement (6.25%) was not satisfied.

Our tools were not well known by the participants, who reported several usability issues that are reflected in the difficulty they perceived during the task of specifying a requirement with MockPlug. In spite of these problems, our approach allowed to participants to be more satisfied. We think that this difference is due to: (1) aid provided by mockup for focusing on specific elements (UI widgets) which resulted in more concrete and clearer specifications for scripters, (2) the possibility of manipulating existing UI components easily, and (3) the possibility of defining behavior for each widget without programming knowledge.

Another additional benefit of using our approach was related to the development process of the scripts. By using MockPlug and MockPlug models, some code could be generated automatically, at least what is related to the fact of creating new widgets and get the existing DOM elements from the target Web page that were manipulated.

5 Related Works

In this section we compare our approach with two kinds of works. First, we analyse other mechanisms and tools to specify UI prototypes. On the other hand, we review some approaches to elicit requirements in the context large crowd of users.

Using UI mockups as a requirement elicitation technique is a growing trend that can be appreciated just observing the amount of different Web and desktop-based prototyping tools like Balsamiq and Mockingbird[8] that appeared during the last years. Statistical studies have proven that mockups use can improve the whole development process in comparison to use other requirements artefacts [15]. Mockups use has been introduced in different and heterogeneous approaches and methodologies. On the one hand, they have been included in traditional, code-based Agile settings as an essential requirement artifact [20] [6]. On the other hand, they have been used as formal specifications in different Model-Driven Development approaches like MockupDD [16], ranging from interaction requirements to data intensive Web Applications modelling. In this work we propose to specify augmentation changes using (among other techniques) mockup-style widgets. Also, MockPlug combines mockup-style augmentation techniques with well known annotations capabilities of common mockup tooling, that also can be used as formal specifications in the future as in [14].

There are new works on collaboration in requirements elicitation. Some authors have used social networks to support requirements elicitation in large-scale systems with many stakeholders [18]. They have used collaborative filtering techniques in order to obtain the most important and significant requirements from a huge number of stakeholders. A more formal and process-centered approach is proposed in [12]. However, we consider that none of the approaches provide a mechanism to collaboratively elicit, describe and validate requirements by using prototype definition for large-scale systems with a large number of stakeholders. The aforementioned works are limited to eliciting narrative requirements. We propose to define requirements by specifying high-fidelity prototypes that are later managed collaboratively from a common repository.

[8] `http://gomockingbird.com`

The ability of creating, validating and evolving prototypes collaboratively just by dragging and dropping widgets over existing Web sites, may allow stakeholders to express their needs in more accurate way, beyond textual descriptions [1]. In the particular context of Web augmentation communities, by improving communication between users and userscripters, we ensure that user requirements are correctly understood before future development of the concrete scripts. In this way, our integrated platform aims to reduce users' efforts when defining their requirements and upload them to the repository where, at the same time, it enables whole the community to improve them iteratively.

In the context of Web augmentation for annotations, there are also many tools to share user's annotations – for instance, Appotate[9] or Diigo[10]. These kinds of tools allow users to share annotations across the web, although these are very useful to share and highlight ideas, annotations are not enough to define complex augmentation requirements.

By using traditional mockups tools users can create their own prototypes, but usually have to start from scratch and widgets are not linked to the Web site where the requirement surfaces. This simple fact would demand much effort from scratch (in order to give some context to the augmentation requirement). Besides that, Web augmentation requirements could be extremely related to existing UI components, and desirably, the prototype should show how these existing components interact with the new widgets. Our approach enables users to quickly define high-fidelity prototypes related to the augmentation requirement, manipulating the same Web page that is going to be transformed by a script in real time. Our MockPlug metamodel makes it possible to share and reproduce these prototypes, but additionally, it also makes possible to process a prototype in order to generate a first script template based on those manipulated widgets.

6 Conclusions and Future Work

It is very complex to gather requirements from a crowd of users for widely used Web applications. In this work we have presented a methodology and its implementation for gathering requirements in the context of Web augmentation communities. Our approach is inspired in agile methodologies were users have an active role. We found that using high-fidelity mockups is very useful for specifying Web augmentation requirements using the running Web page as a foundation for specifying them graphically. With *CrowdMock*, users may define their own requirements and share them with the community, who can reproduce, edit and evolve them collaboratively. CrowdMock and its tools have been evaluated obtaining some benefits in comparison with traditional approaches. By using the proposed high-fidelity mockups that run over existing Web pages, we agilize the definition process since it is not necessary to construct an abstract conceptualization from existing UI. Users know what they want from Web applications and we try to give them the tools and mechanisms for naturally expressing their requirements.

[9] `http://appotate.com`
[10] `https://www.diigo.com`

We described the details of an evaluation of CrowdMock approach, which showed some positive results in Web augmentation requirements gathering and also some usability issues in the MockPlug tool that we are currently addressing. Integration between MockPlug and UserRequirements is also being improved. Additionally, we have an ongoing work on the server-side application (UserRequirements), which is being enhanced to support better collaboration mechanisms.

Another interesting further work path includes testing support and better code generation. We are planning to automate test for userscripts. Code generation is currently made without considering good *userscripters* practices and patterns; thus, the generated code may be not be easy to understand or manually adapt by userscripters. Thus, we are working on improvements our MockPlug code generators API in order to allow *userscripters* to develop their own code generators, which are a particular type of MockPlug plug-ins.

Although we are focused on Web augmentation requirements, our approach could also be useful for projects intended to build software products from scratch instead of augment existing ones. Thus, we are planning to improve the expressiveness of our metamodel, extending the tool with new kind of widgets and finally improve the server-side repository with features related to general developer support.

References

1. Brooke, J.: SUS: A 'quick and dirty' usability scale. In: Usability Evaluation in Industry, pp. 189–194. Taylor and Francis, London (1996)
2. Brusilovsky, P., Kobsa, A., Nejdl, W.: The Adaptive Web (2008)
3. Cameron Jones, M., Churchill, E.: Conversations in Developer Communities: a Preliminary Analysis of the Yahoo! Pipes Community. In: Proceedings of the Fourth International Conference on Communities and Technologies, pp. 195–204. ACM (2009)
4. Cohn, M.: User stories applied: for agile software development, p. 268. Addison-Wesley (2004)
5. Díaz, O.: Understanding Web augmentation. In: Grossniklaus, M., Wimmer, M. (eds.) ICWE Workshops 2012. LNCS, vol. 7703, pp. 79–80. Springer, Heidelberg (2012)
6. Ferreira, F., Noble, J., Biddle, R.: Agile Development Iterations and UI Design. In: Proceedings of AGILE 2007 Conference, pp. 50–58 (2007)
7. Gauch, S., Speretta, M., Chandramouli, A., Micarelli, A.: User Profiles for Personalized Information Access. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) Adaptive Web 2007. LNCS, vol. 4321, pp. 54–89. Springer, Heidelberg (2007)
8. Homrighausen, A., Six, H., Winter, M.: Round-Trip Prototyping Based on Integrated Functional and User Interface Requirements Specifications. Requir. Eng. 7(1), 34–45 (2002)
9. Hussain, Z., Slany, W., Holzinger, A.: Current state of agile user-centered design: A survey. In: Holzinger, A., Miesenberger, K. (eds.) USAB 2009. LNCS, vol. 5889, pp. 416–427. Springer, Heidelberg (2009)
10. Kelly, S., Tolvanen, J.P.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society (2008)

11. Ko, A., Abraham, R., Beckwith, L., Blcakwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M., Rothermel, G., Shaw, M., Wiedenbeck, S.: The State of the Art in End-User Software Engineering. ACM Computing Surveys, 1–44 (2011)
12. Konaté, J., El Kader Sahraoui, A., Kolfschoten, G.: Collaborative Requirements Elicitation: A Process-Centred Approach. Group Decision and Negotiation Journal (2013)
13. Kulak, D., Guiney, E.: Use cases: requirements in context. Addison-Wesley (2004)
14. Mukasa, K., Kaindl, H.: An Integration of Requirements and User Interface Specifications. In: Proceedings of 6th IEEE International Requirements Engineering Conference, pp. 327–328 (2008)
15. Ricca, F., Scanniello, G., Torchiano, M., Reggio, G., Astesiano, E.: On the effectiveness of screen mockups in requirements engineering. In: Proceedings of ACM-IEEE Int. Symp. Empir. Softw. Eng. Meas. ACM Press, New York (2010)
16. Rivero, J.M., Rossi, G.: MockupDD: Facilitating Agile Support for Model-Driven Web Engineering. In: Sheng, Q.Z., Kjeldskov, J. (eds.) ICWE Workshops 2013. LNCS, vol. 8295, pp. 325–329. Springer, Heidelberg (2013)
17. Schneider, K.: Generating fast feedback in requirements elicitation. In: Proceedings of the 13th International Working Conference on Requirements Engineering: Foundation for Software Quality, pp. 160–174 (2007)
18. Lim, S.L., Finkelstein, A.: StakeRare: Using Social Networks and Collaborative Filtering for Large-Scale Requirements Elicitation. IEEE Transactions on Software Engineering 38(3), 707–735
19. Sutherland, J., Schwaber, K.: The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process, http://assets.scrumfoundation.com/downloads/2/scrumpapers.pdf?1285932052 (accessed: February 16, 2014)
20. Ton, H.: A Strategy for Balancing Business Value and Story Size. In: Proceedings of AGILE 2007 Conference, pp. 279–284 (2007)
21. UserScripts, http://userscripts.org (accessed: February 16, 2014)
22. Yahoo Pipes, http://pipes.yahoo.com/pipes/ (accessed: February 16, 2014)
23. Yu, J., Benatallah, B., Casati, F., Florian, D.: Understanding mashup development. IEEE Internet Computing 12(5), 44–52 (2008)

An Empirical Study on Categorizing User Input Parameters for User Inputs Reuse

Shaohua Wang[1], Ying Zou[2], Bipin Upadhyaya[2], Iman Keivanloo[2], and Joanna Ng[3]

[1] School of Computing, Queen's University, Kingston, Ontario, Canada
shaohua@cs.queensu.ca
[2] Electrical and Computer Engineering, Queen's University, Kingston, Canada
{ying.zou,bipin.upadhyaya,iman.keivanloo}@queensu.ca
[3] CAS Research, IBM Canada Software Laboratory, Markham, Ontario, Canada
jwng@ca.ibm.com

Abstract. End-users often have to enter the same information to various services (*e.g.*, websites and mobile applications) repetitively. To save end-users from typing redundant information, it becomes more convenient for an end-user if the previous inputs of the end-user can be pre-filled to applications based on end-user's contexts. The existing pre-filling approaches have poor accuracy of pre-filling information, and only provide limited support of reusing user inputs within one application and propagating the inputs across different applications. The existing approaches do not distinguish parameters, however different user input parameters can have very varied natures. Some parameters should be pre-filled and some should not. In this paper, we propose an ontology model to express the common parameters and the relations among them and an approach using the ontology model to address the shortcomings of the existing pre-filling techniques. The basis of our approach is to categorize the input parameters based on their characteristics. We propose categories for user inputs parameters to explore the types of parameters suitable for pre-filling. Our empirical study shows that the proposed categories successfully cover all the parameters in a representative corpus. The proposed approach achieves an average precision of 75% and an average recall of 45% on the category identification for parameters. Compared with a baseline approach, our approach can improve the existing pre-filling approach, *i.e.*, 19% improvement on precision on average.

Keywords: User Input Parameters Categories, User Inputs Reuse, Auto-filling, Ontology, Web Forms.

1 Introduction

Web is becoming an essential part of our daily life. Various on-line services (*e.g.*, web services and mobile applications) allow end-users to conduct different web tasks, such as on-line shopping and holiday trip planning. These services require end-users to provide data to interact with them and some of the data provided by end-users are usually repetitive. For example the AVIS[1], a car rental website, and the Homes [2], a real estate website, require end-users to enter their personal information such as *First Name* and *Last Name* illustrated in Figure 1 and Figure 2. It could be a cumbersome and annoying process for an end-user, especially a smartphone end-user, to fill the same information into web forms with multiple input fields repeatedly. Therefore information pre-filling becomes critical to save end-users from this cumbersome process. Rukzio et al. [1] found that end-users are four times faster on smartphones when they just have to correct pre-filled form entries compared to entering the information from scratch. A web form within web or mobile applications usually consists of a set of input fields assigned with a label, for example the input field *Contact Phone Number* in Figure 1. The label of this input field is "Contact Phone Number", the type of this input field is *text field*. Throughout this paper, we consider an input field as a user input parameter and an end-user input (*i.e.*, a piece of information) as the value which can be filled into an input field.

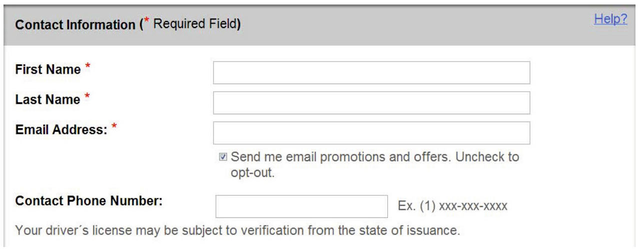

Contact Information (* Required Field)
Help?
First Name *
Last Name *
Email Address: *
☑ Send me email promotions and offers. Uncheck to opt-out.
Contact Phone Number: Ex. (1) xxx-xxx-xxxx
Your driver´s license may be subject to verification from the state of issuance.

Fig. 1. A sample screen shot of a web form requiring user's personal information to reserve a car

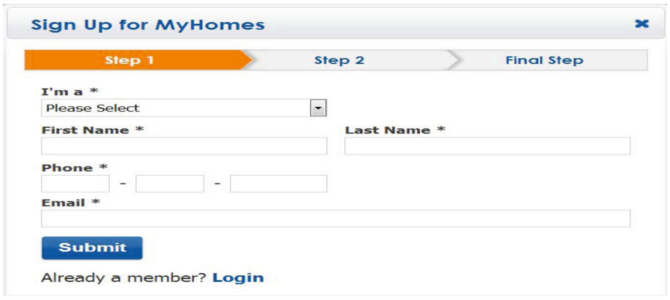

Sign Up for MyHomes
Step 1 Step 2 Final Step
I'm a *
Please Select
First Name *
Last Name *
Phone *
Email *
Submit
Already a member? Login

Fig. 2. A sample screen shot of a web form requiring user's personal information to sign up the website

Recently, several industrial tools and academic approaches have been developed to pre-fill user inputs into web forms automatically. Web browsers usually

[1] `http://www.avis.ca/car-rental/avisHome/home.ac`
[2] `http://www.homes.com/`

provide web form Auto-filling tools, such as Firefox form auto-filling Addon [2] and Chrome form auto-filling tool [3] to help users fill in forms. In general, the auto-filling tools record the values entered by a user for specific input fields pre-configured by the tools in a given form. Then, they identify the input fields which are identical or similar to the pre-configured input fields by string matching, and fill the entered values into the identified inputs fields when the users visit websites. Since such approaches are error-prone, they allow end-users to modify the values. Recently, a few academic studies such as [4][5][6][7] proposed several approaches to help end-users. Hartman et al. [4] propose a context-aware approach using user's contextual information to fill in web forms. Toda et al. [5] propose a probabilistic approach using the information extracted from data-rich text as the input source to fill values to web forms. Wang et al. [6] propose an approach using the relations between similar input parameters to fill values to web forms. Araujo et al. [7] extract the semantic of concepts behind the web form fields and use the relations of concepts to fill in web forms. However, all these tools and approaches suffer from two main drawbacks:

– **Poor accuracy of pre-filling values to input parameters of web forms or applications**. The pre-filled information can be out of date and out of context. The user often has to modify the pre-filled values passively. With the rapid growth of the population of mobile application users, it is even more frustrating for mobile application users to modify the pre-filled values due to the restrictions in screen size and typing. Accurate pre-filling values becomes a critical step to enhance the user experience.
– **Limited support of reusing user inputs within an application and propagating them across different applications.** The existing methods can only reuse a few types of user inputs within an application or across different applications. For example an end-user is planning a holiday trip from Toronto to Miami, he or she could conduct two tasks of searching for cheap flight tickets and cheap transportation from airport to hotel. The end-user needs browse different websites to find the most appropriate solution for him or her. During the process of conducting these two tasks, the similar or identical input parameters from different websites or within a website, such as departure and return cities, should be linked together, and reuse user inputs among them. In this scenario, for example, Chrome form auto-filling tool [3] cannot help the end-users, since it can only reuse basic personal information such as credit card information and addresses.

The existing tools and approaches treat all the user input parameters equally, and the matching mechanism of existing approaches is only based on the string matching or semantic similarity calculation of field names. If a match is identified, the existing approaches pre-fill a value to an input parameter. However different user input parameters can have very varied natures. For example, the coupon number is only valid for a single purchase, therefore may not be suitable for pre-filling. The input fields for departure and destination cities from a flight booking website should not be pre-filled with previous user inputs without knowing user's context, the end-user's name can be pre-filled, since the end-user's name does not change in most cases.

It is crucial to improve the accuracy of automatic value pre-filling by understanding the characteristics of different user input parameters. In this paper, we propose an ontology model for expressing common parameters and approach based on the proposed ontology model to automatically identify a category for an input parameter. We conducted two empirical studies. The first empirical study was served as an initial empirical investigation to explore the characteristics of user input parameters. The first empirical study was conducted on 30 popular websites and 30 Android mobile applications from Google Play Android Market [3]. Based on the results of our first study, we identify and propose four categories for input parameters from web and mobile applications. The proposed categories offer a new view for understanding input parameters and provide tool developers guidelines to identify pre-fillable input parameters and the necessary conditions for pre-filling. To the best of our knowledge, we are the first to propose categories for input parameters to explore the characteristics of user input parameters. We conducted the second empirical study to verify the effectiveness of proposed categories and approach for category identification. The second study was conducted on 50 websites from three domains and 100 mobile applications from five different categories in Google Play Android Market.

The major contributions of our paper are listed as follows:

- We propose four categories of user input parameters to capture the nature of different user input parameters through an empirical investigation. For pre-filling, each category of parameters should be collected, analyzed and reused differently. The results of our empirical study show that our categories are effective to cover all the input parameters in a representative corpus.
- We propose an ontology model to express the common parameters and the relations among them. Moreover, we use the proposed ontology model to carry the category information for input parameters. We propose a WordNet-based approach that automatically updates the core ontology for unseen parameters from new applications. The results of our empirical study show that our approach obtains a precision of 88% and a recall of 64% on updating the ontology to include the unseen parameters on average.
- We propose an ontology-based approach to identify a category for input parameters automatically. On average, our approach for category identification can achieve a precision of 90.5% and a recall of 72.5% on the identification of a category for parameters on average.
- We test the effectiveness of our proposed categories on improving the existing approaches. We build a baseline approach which does not distinguish the different characteristics of input parameters, and incorporate our proposed categories with the baseline approach to form a category-enabled approach. We compare two approaches through an empirical experiment. The results show that our approach can improve the baseline approach significantly, *i.e.*, on average 19% in terms of precision.

The rest of the paper is organized as follows. Section2 describes the user interfaces of web and mobile applications. Section 3 presents our proposed approach

[3] `https://play.google.com/store?hl=en`

for categorizing input parameters. Section 4 introduces the empirical studies. Section 5 discusses the threats to validity. Section 6 summarizes the related literature. Finally, section 7 concludes the paper and outlines some avenues for future work.

2 Web and Mobile Application User Interface

In this paper, we study the user input parameters from the user interfaces of web applications and mobile applications.

Web pages are mainly built with HTML and Cascading Style Sheets (CSS), and can be converted into an HTML DOM tree [4]. A Web form is defined by an HTML FORM tag <form> and the closing tag *</form>*. A web form usually consists of a set of input elements, such as the text fields in Figure 2, to capture user information. An input element is defined by an HTML INPUT tag <input> specifying an input field where the user can enter data. The input element can contain several attributes, such as *name* specifying the name of an <input> element, *type* defining the type <input> to display (*e.g.*, displayed as a button or checkbox) and *value* stating the value of an <input> element. An <input> element can be associated with a human-readable label, such as First Name. This information can be accessible by parsing HTML source code.

There are three types of mobile applications:

– *Native Apps:* The native apps are developed specifically for one platform such as Android[5], and can access all the device features such as camera.
– *Web Apps:* The web apps are developed using standard Web technologies such as Javascript. They are really websites having *look and feel* like native apps. They are accessed through a web browser on mobile devices.
– *Hybrid Apps:* The hybrid apps are developed by embedding HTML5 apps inside a thin native container.

An Android application is encapsulated as an Android application package file (APK) [6]. An APK file can be decoded into a nearly original form which contains resources such as source code (*e.g.*, Java) and user interface layout templates in XML files that define a user interface page or a user interface component if it is not a HTML5 application. In this study, we only study the native mobile applications because the user interface templates in XML files can be obtained by decoding APK files.

3 Our Proposed Approach for Categorizing User Input Parameters

In this section, we first present an ontology model for expressing common user input parameters and their relations. Second, we propose an automatic approach for updating the ontology. Third, we propose an ontology-based approach for categorizing input parameters.

[4] `http://www.w3schools.com/htmldom/dom_nodes.asp`
[5] `http://www.android.com/`
[6] `http://en.wikipedia.org/wiki/APK_(file_format)`

3.1 An Ontology Definition Model

In this study, we build an ontology to capture the common user input parameters and the five relations among parameters. Figure 3 illustrates the main components of ontology definition model and their relations.

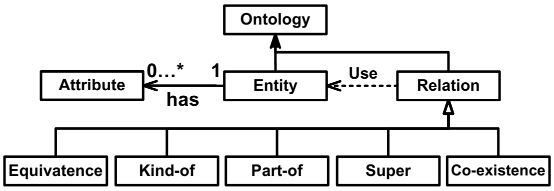


Fig. 3. Components of ontology definition model

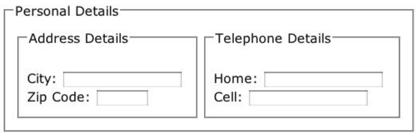


Fig. 4. An example of 4 user inputs parameters: City, Zip Code, Home and Cell

The components are listed as follows:

- *Entity:* is an input parameter from a website or mobile application, or a concept description of a group of resources with similar characteristics.
- *Attribute:* is a property that a parameter can have. There are four attributes:
  - *Category.* The category defines the category information of a parameter.
  - *Label.* The category stores the *Label* (*i.e.*, a human-readable description) of an input field from websites or mobile applications.
  - *Coding Information.* The category stores the HTML coding information (*i.e.*, websites) or XML templates (*i.e.*, mobile applications).
  - *Concepts.* The category stores the concepts related to the parameter.
- *Relation:* defines various ways that entities can be related to one another. The five relations are listed as follows:
  - *Equivalence.* Two entities have the same concept such as Zip Code and Postal Code.
  - *Kind-of.* One entity is a kind-of another one. For example, Home (*i.e.*, home phone) is a kind of Telephone Details illustrated in Figure 4.
  - *Part-of.* One entity is a part-of another one, such as Zip Code and Address Details illustrated in Figure 4.
  - *Super.* One entity is a super of another. The super relation is the inverse of the Kind-of relation. For example, Telephone Details is a super of Home (*i.e.*, home phone) illustrated in Figure 4.

- *Co-existence.* Two entities are both a part-of an entity and they are not in the relation of equivalence, such as City and Zip Code illustrated in Figure 4.

Usually the user input parameters are terminal concepts [8], such as *Phone Number* and *Price*, which have no sub-concepts. During the process of ontology creation, we use the UI structure and semantic meanings of the parameters to identify the relations. If two input elements have the same parent node in an HTML DOM tree, their relation is co-existence, and the relation between the two input parameters and their parent HTML DOM node is part-of. For example, the user input parameters *City* and *Zip Code* co-exist and they have a part-of relation with Address Details in Figure 4. Figure 5 shows the visualization of the ontology that is built based on the example in Figure 4.

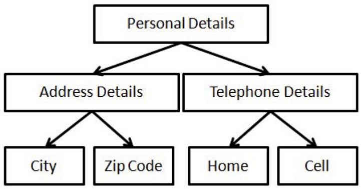


Fig. 5. An example ontology of personal details containing four user inputs parameters: City, Zip Code, Home and Cell

3.2 Our Approach of Updating Ontology

Once the initial ontology based on the proposed ontology model (Section 3.1) is established, an automatic approach for updating the ontology is required to add a new parameter into the ontology. We use WordNet [9], a large lexical database for English and containing semantic relations between words, to identify the relations between the new parameter and the existing ones in the ontology. The following relations of words defined in WordNet are used to identify our 5 relations:

1. If two words have the same synsets, they have a same semantic meaning, we convert it to Equivalence relation.
2. **Hypernym** shows a kind-of relation. For example, car is a hypernym of vehicle. We convert it to a kind-of relation.
3. **Meronym** represents a part-whole relation. We convert it to a part-of relation.
4. **Hyponym** defines that a word is a super name of another. We convert it to super relation. Hyponym is the inverse of hypernym meaning that a concept is a super name of another. For example, vehicle is a hyponym of car.
5. We use the part-of relation to identify the co-existence relation. If a word with another word both have a part-of relation with a same word, this word and the other word have co-existence relation.

3.3 Our Approach for Category Identification

It is important for a pre-filling approach to know the category of a user input parameter automatically. In this section, we introduce our ontology-based approach for identifying a category of an input parameter in details. Our approach uses the proposed ontology definition model proposed in Section 3.1. Our approach uses two strategies which are listed as follows:

- *Concept-based Strategy.* This strategy relies on an ontology of user input parameters to identify a category for a user input parameter automatically. If two input parameters have the same concepts, these two input parameters belong to the same category.
- *UI-based Strategy.* This strategy relies on the design of the UI layouts. We consider two user input parameters are the nearest neighbours to each other if their input fields are contained in the same parent UI component. Figure 4 shows an example of 4 user input parameters: *City, Zip Code, Home and Cell.* The *City and Zip Code* are the nearest neighbours to each other since they have the same parent UI node which has a label *Address Details.* We assume that if all the neighbours of a user input have been categorized and belong to the same category, there exists a high chance that they belong to the same category.

Given an ontology having a set of parameters, which is denoted as O $= < P_1^O, \ldots, P_n^O >$, where n is the number of parameters, and an input parameter P, our approach uses the following steps to identify a category for P:

- Step 1. We use WordNet synsets to retrieve synonyms of the concepts of the parameter P to expand the concept pool (*i.e.*, a bag of words) of P and the concept pool of each P_i^O, where $1 \leq i \leq n$.
- Step 2. We identify the P_j^O (where $1 \leq j \leq n$) whose concept pool has the concepts which are the synonyms of (or identical to) any concepts in the concept pool of P (*i.e.*, having the same semantic meaning).
- Step 3. If multiple parameters in the ontology are identified for P in Step 2, we choose the parameter sharing the most common concepts with the concept pool of P as the identical parameter to P. We assign the category of the chosen parameter to P
- Step 4. If no parameter is identified in Step 2, we apply the UI-strategy on the given ontology O and input parameter P. We identify the neighbors of P from its UI and repeat Step 1-3 to identify a category for every neighbor. If any neighbor of P cannot be categorized (*i.e.*, no parameters in the ontology have the same concepts as the neighbor does) or the neighbors of P have different categories, we cannot categorize P. If all the neighbors of P belong to a same category, we assign this category to input parameter P.

4 Empirical Study

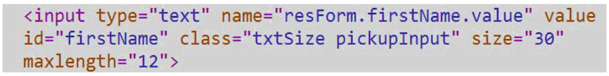
In this study, we conduct two studies on different datasets. The first study is designed to study the different characteristics of parameters and propose categories for input parameters. The second empirical study is designed to evaluate the effectiveness of the proposed categories and the approach for categorizing input parameters.

4.1 First Empirical Study

We conduct an initial study to understand the nature of user input parameters and categorize the parameters. Understanding the different characteristics of parameters of different categories can help analyze, process and pre-fill the parameters differently to improve the accuracy of pre-filling. In this section, we introduce the study setup, the analysis method and the findings of our empirical investigation.

4.1.1 Data Collection and Processing

The study is conducted on 30 websites (*i.e.*, 15 shopping websites and 15 travel planning websites) and 30 mobile applications (*i.e.*, 15 shopping apps and 15 travel apps). We collect the user input parameters from web and mobile applications in different ways.



```
<input type="text" name="resForm.firstName.value" value
id="firstName" class="txtSize pickupInput" size="30"
maxlength="12">
```

Fig. 6. A sample screen shot of source code of an input field

Collecting input parameters from websites: First, we manually visit the website to bypass the login step. Second, we use *Inspect Element*[7] of Google Chrome[8] to collect the following information of a user input parameter:

– **Label:** The value of the label (*i.e.*, the descriptive text shown to users).
– **Attributes:** The values of the attributes of the input element such as id and name. For example, Figure 6 shows the source code of the input field First Name in Figure 1.

Collecting input parameters from mobile applications: Instead of running each mobile application, we use Android-apktool [9], a tool for reverse engineering Android APK files, to decode resources in APKs to readable format. We build a tool to extract the information of a form from UI layout XML files if exist, then collect the information related to a user input parameter in the same way as we collect from websites.

Data cleaning: The extracted information need to be cleaned for further processing. For example, the extracted information for the input parameter *First Name* in Figure 1 is {First Name, resForm.firstName.value, FirstName, text} need to be cleaned. We remove the duplicated phrases and programming expressions. For the given example above, the output after cleaning is {First Name, res, Form, value, text}.

[7] `https://developers.google.com/chrome-developer-tools/docs/elements`
[8] `https://www.google.com/intl/en/chrome/browser/`
[9] `https://code.google.com/p/android-apktool/`

Data processing: We manually process the information of parameters as follows: First, we identify the concepts from the information of a user input parameter. A concept is a semantic notion or a keyword for describing a subject (*e.g.*, "taxi" and "city"). Second, the parameters having the same concepts or same semantical meanings are merged and considered as one parameter which we save all the unique concepts for. For example, two input parameters from two different flight searching websites, one with the label *Departure* and the other one with the label *Leave* should be merged and considered as one parameter having two concept words *Departure* and *Leave*.

4.1.2 Results of First Empirical Study

We collected 76 and 32 unique user input parameters from websites and mobile applications respectively. Due to the fact that the user input parameters are repeated among different applications, we observe that the number of unique parameters identified decreases with the increase of the number of applications studied. The 76 parameters extracted from the websites (*i.e.*, shopping and travel) contain all the 32 parameters from mobile applications from the same domains. This is due to the fact that mobile applications usually are the simplified versions of corresponding websites.

After studied the user input parameters, we found that input parameters can be categorized into four categories. The four categories are listed as follows:

– *Volatile Parameters:* This type of parameters have no change of state. They can be further categorized into two sub-categories:
  - One-Time Parameters: The values of this type of parameters are valid for one interaction, such as coupon number. This type of parameters should not be used for pre-filling at all.
  - Service-Specific Parameters: The values of this type of parameters can only be used for a specific service (*e.g.*, a website or a mobile application). For example, Member ID of the BESTBUY Reward Zone[10] in Canada can only be used for "Sign In" function or "Account Set Up" function of reward service. When end-users receive a membership card, they need enter the Member ID to set up an account in BESTBUY Reward Zone. This type of parameters can be pre-filled, however the parameters cannot be reused by different services.

– *Short-time Parameters:* The values of this type of parameters change with high frequency. For example during the course of conducting an on-line clothes shopping, an end-user may use the different colors as the search criteria to find the best suitable clothes from different services for the user, during a short period of time, the value of the color can be pre-filled. This type of parameters can be pre-filled and reused by different services, however it is extremely hard to pre-fill this type of parameters unless some conditions are met. For example, in the above example of searching clothes, the value of the color should be pre-filled only if the end-user has not switched to a new task.

[10] `https://www.bestbuyrewardzone.ca/`

- *Persistent Parameters:* The values of this type of parameters do not change over a long period of time. For example, the gender of the end-user and permanent home address. This type of parameters are suitable for pre-filling and being reused across different services.
- *Context-aware Parameters:* There exist some user input parameters which are context-dependent, such as the user's location and current local time. This type of parameters can be pre-filled based on user's contextual information. The value of a context-aware parameter can be obtained from two types of data sources:
  - *Direct Data Sources.* The context data is directly available and accessible with little computation, such as entries in Google calender, time from mobile phone clock, "my" To-do lists from task manager and a friend's preferences on Facebook.
  - *Analyzed Data Sources.* With the accumulation of user history, additional contextual data can be mined from history data such as user behaviors. For example, a user always books a round-trip business class flight ticket from Toronto to Los Angeles for business trips and a round-trip economic class flight ticket from Toronto to Vancouver for personal trips.

  In this paper, we consider only the user input parameters whose values can be obtained directly from available sources.

4.2 Second Empirical Study

The goals of the second empirical study are to: 1) examine the representativeness of the proposed categories of parameters; 2) validate the effectiveness of the approach of updating ontology; 3) test the effectiveness of the proposed approach for category identification; 4) validate the effectiveness of our proposed categories on improving existing approaches of pre-filling values to web forms.

4.2.1 Data Collection and Processing

We conduct our second empirical study on 45 websites from three different domains: Finance, Healthcare and Sports (*i.e.*, 15 websites from each domain) and 100 Android mobile Applications from five different categories: Cards& Casino, Personalization, Photography, Social and Weather in Google Play Market (*i.e.*, 20 applications from each studied category). The studied websites and mobile applications are different from the ones studied in our first empirical study. We use the same approach as discussed in section 4.1.1 to collect user input parameters from web and mobile applications. In total, there are 146 and 68 unique user input parameters from web and mobile applications respectively.

4.2.2 Research Questions

We presents four research questions. For each research question, we discuss the motivation of the research question, analysis approach and the corresponding findings.

RQ1. Are the proposed categories of parameters sufficient to cover the different types of user input parameters?

Motivation. The existing tools and approaches do not distinguish the characteristics of user input parameters. The parameters are processed equally. Therefore the lack of knowledge on the user input parameters negatively impacts the accuracy of pre-filling approaches, and is the main reason for limited support of reusing parameters within one application or across multiple applications. Categorizing the users input parameters can help us in understanding the different characteristics of parameters. Each category of parameters has its unique characteristics which need to be fully understood.

In Section 4.1, we identified 4 categories for user input parameters via a manual empirical study. In this research question, we validate the proposed categories to see if they can explain the further unseen user input parameters collected in the empirical study.

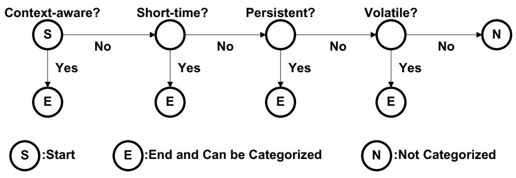


Fig. 7. The decision tree for the judgment process

Analysis Approach. To answer this question, we study the user input parameters collected from the 45 websites and 100 mobile apps. For each user input parameter, we manually process it to see whether a user input parameter belongs to any category or not. The judgment process is based on a decision tree as shown in Figure 7. When we process a parameter, we first decide whether it is a context-aware parameter or not. If no, we further decide whether the parameter can be categorized as a short-time parameter or not. If no, then we decide whether the parameter is a persistent parameter or not. If no, finally we decide whether the parameter is a volatile parameter or not. If it is still a no, the parameter cannot be categorized by using our proposed categories of input parameters.

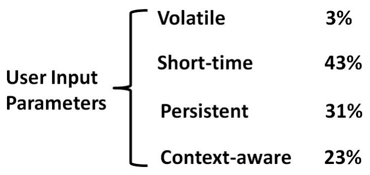


Fig. 8. The percentage of each category

Results. Figure 8 shows that all of the unseen user inputs parameters can be categorized into our proposed categories and there is no "not Categorized". More specifically, 43% of the parameters are short-time and only 3% of the parameters are volatile parameters. The results suggest that our categories are enough to describe all the unique user input parameters in our empirical study.

RQ2. Is the proposed approach for updating ontology effective?

Motivation. Usually the ontologies are created manually by web filling tool builders or contributors from the communities of knowledge sharing. It is helpful to have an automatic approach to update and expand the ontologies. We test the effectiveness of the WordNet-based approach of updating ontology proposed in Section 3.2.

Analysis Approach. We conduct two experiments to answer this question. In the first experiment, our approach for updating ontology uses the input parameters from one domain of applications to update the ontology to include the parameters from other domains of applications. In the second experiment, our approach for updating ontology uses the parameters from one domain of applications to update the ontology to include the parameters from the same domain of applications.

To conduct *experiment 1*, we construct an ontology[11], denoted as $O^{Initial}$, using the 76 user inputs parameters from the first empirical study (Section 4.1). Second, we use the ontology $O^{Initial}$ as the existing ontology to include the parameters of 146 parameters from web applications and 68 parameters from mobile applications (Section 4.2.1). Third, we compute the precision and the recall of the approach using Equation (1) and Equation (2) respectively. The precision metric measures the fraction of retrieved user input parameters that are placed in the correct place (*i.e.*, having the right relations with other parameters), while the recall value measures the fraction of correct user input parameters that are retrieved.

$$precision = \frac{|\{correct\ Parameters\} \bigcap \{retrieved\ Parameters\}|}{|\{retrieved\ Parameters\}|} \tag{1}$$

$$recall = \frac{|\{correct\ Paras\} \bigcap \{retrieved\ Paras\}|}{|\{correct\ Paras\}|} \tag{2}$$

The goal of *experiment 2* is to see whether we can obtain a better result if we build a domain dependent ontology for each domain in our empirical study. To conduct *experiment 2*, we conduct the following steps on the 45 websites:

1. randomly select 5 of 15 Finance websites and construct the domain specific ontology for the user input parameters in Finance using the user input parameters from the selected websites.
2. use the domain dependent ontology of parameters in Finance as an existing ontology and perform a manual updating on the existing ontology to include the user input parameters from the rest of 10 Finance websites.

[11] `https://dl.dropboxusercontent.com/u/42298856/icwe2014.html`

3. apply the automatic ontology updating approach on the parameters from the rest of 10 Finance websites to verify its performance.

We replicate the previous steps on 15 Health Care websites and 15 Sports websites. We conduct our analysis on mobile applications in the same way as we do on websites, the only difference is that we randomly select 5 mobile applications from each domain. We compute the precision and the recall of the approach using Equation (1) and Equation (2) respectively.

Table 1. Results of our approach for updating ontology using domain-specific ontology

Type	Domain	Precision(%)	Recall(%)
website	Finance	83	66
website	Health Care	78	69
website	Sports	95	65
mobile	Cards&Casino	92	74
mobile	Personalization	87	48
mobile	Photography	90	62
mobile	Social	85	42
mobile	Weather	97	85

Results. In the experiment 1, our approach for updating existing ontology obtains a precision of 74% and a recall of 42% on websites, and a precision of 82% and a recall of 30% on mobile applications. We further investigate our results to gain further insights regarding the low recalls. We found that 35 of 146 website user input parameters and 25 of 68 mobile applications user input parameters can be found in the ontology constructed using the input parameters collected during the initial empirical investigation. The low recalls indicate that it is hard to use one general ontology as the existing ontology to include the parameters from other domains automatically, although the precisions are considerably high.

In experiment 2, Table 1 shows that our approach for updating ontology obtain a precision of 85% and a recall of 67% on websites, and a precision of 90% and a recall of 62% on mobile applications. The results of our approach in **experiment 1** can be improved by using the domain-specific ontology as the existing ontology. On average, the approach can be improved by 11% in terms of precision and 25% in terms of recall on websites, and 8% in terms of precision and 32% in terms of recall on mobile applications.

RQ3. Is our approach of category identification effective?

Motivation. After showing the representativeness of our categories for input parameters, we propose an ontology-based approach to identify a category for an input parameter. In this section, we investigate the effectiveness of our approach for identifying a category for an input parameter.

Analysis Approach. To assess the performance of the proposed approach, we proceed as follows:

1. We use the domain-specific ontologies constructed in RQ2 as the training ontologies.
2. We categorize the parameters in ontologies by assigning a category to each parameter.
3. We locate the nearest neighbours for the input parameters which are not in the training ontologies.
4. We apply our approach on the parameters in Step 3

We compute the precision and the recall of the approaches using Equation (1) and Equation (2) respectively. The precision metric measures the fraction of retrieved user input parameter that are categorized correctly, while the recall value measures the fraction of correct user input parameters that are retrieved.

Table 2. Results of our approach of category identification

Type	Domain	Precision(%)	Recall(%)
website	Finance	90	79
website	Health Care	92	82
website	Sports	89	70
mobile	Cards&Casino	90	78
mobile	Personalization	91	58
mobile	Photography	95	62
mobile	Social	77	55
mobile	Weather	100	88

Results. Table 2 shows the precision and recall values of our approach to identify a category for input parameters. On average, our approach can achieve a precision of 90% and a recall of 77% on websites, and a precision of 91% and a recall of 68% on mobile applications. Our approach works well on the input parameters from mobile applications in the domain of weather, because usually the weather mobile apps relatively have fewer number of input parameters and very similar functionalities.

RQ4. Can the proposed categories improve the performance of pre-filling?

Motivation. The existing pre-filling approaches (*e.g.*, [7][14]) treat all the input parameters equally. They do not take into consideration the characteristics of input parameters. These approaches are usually based on the string matching or semantic similarity calculation between the user inputs and the labels of input parameters (*e.g.*, input fields). Essentially they fill a value to an input field as long as a match between a user input and an input field is identified, however the value required by the input parameter may not be suitable for pre-filling due to curtain conditions as mentioned in Section 4.1.2. In this research question, we evaluate the effectiveness of our categories on improving the existing techniques of reusing user inputs.

Analysis Approach. To answer this question, we build a baseline approach which adopts the approach in [7]. The baseline approach [7] pre-fills values to input parameters based on the semantic similarity between user inputs and textual information (*e.g.*, words mined from labels and the attributes of HTML DOM elements defining the input parameters in the user interface) of input parameters. The baseline approach calculates the similarity between user inputs and labels of input parameters using WordNet [9]. The stopword removal, word stemming and non-English words removal are conducted on the textual information of input parameters to identify meaningful words. Then, WordNet is used to expand each word with synsets (*i.e.*, a set of words or collation synonyms) to retrieve synonyms of the found terms.

We build our approach by enriching the baseline approach with the proposed categories for input parameters (Baseline + Categories). We evaluate the improvement of applying our categories with the baseline approach. Our approach use the ontology-based approach proposed in Section 3.3 to identify a category for an input parameter to see whether the input parameter is suitable for pre-filling or not. If the input parameter is suitable for being pre-filled, our approach pre-fills an input parameter. We compute precision using Equation (1) and recall using Equation (2) to measure the improvement. The precision metric measures the fraction of retrieved user input parameters that are pre-filled correctly, while the recall value measures the fraction of correct user input parameters that are retrieved.

To pre-fill input parameters using the user previous inputs, we collect user inputs through our input collector [6] which modifies the Sahi[12] tool to track the user's Web activities. The first author of this paper used the tool to track his inputs on web forms from three domains: Finance, Health and Sports. We apply both the baseline approach and our approach on the 45 websites.

Table 3. Results of the evaluations of our categories on improving the baseline approach of filling input parameters

Domain	Baseline		Baseline+Categories	
	Precision(%)	Recall(%)	Precision(%)	Recall(%)
Finance	50	34	64	36
Health	41	22	67	30
Sports	36	16	53	20

Results. Table 3 shows that our approach incorporating categories of input parameters can improve the baseline approach on average 19% in terms of precision. We further inspect the results and found that our approach can reduce the number of wrong filled values compared with the baseline approach. Some of

[12] `http://sahi.co.in/`

the wrong filled values should not be pre-filled to the input parameters in the fist place.

5 Threats to Validity

This section discusses the threats to validity of our study following the guidelines for case study research [10].

Construct validity threats concern the relation between theory and observation. In this study, the construct validity threats are mainly from the human judgment in categorizing the parameters and ontology construction. We set guidelines before we conduct manual study and we paid attention not to violate any guidelines to avoid the big fluctuation of results with the change of the experiment conductor.

Reliability validity threats concern the possibility of replicating this study. We attempt to provide all the necessary details to replicate our study. The websites and mobile apps we used are publicly accessible[13].

6 Related Work

In this section, we summarize the related work on form auto-filling approaches.

Several industrial tools have been developed to help users fill in the Web forms. Web browsing softwares provide Web form Auto-filling tools (*e.g.*, Firefox form auto-filling addon [2] and Chrome form auto-filling tool [3]) to help users fill in forms. RoboForm [11] is specialized in password management and provides form auto-filling function. Lastpass [12] is an on-line password manager and form filler. 1Password [13] is a password manager integrating directly into web browsers to automatically log the user into websites and fill in forms. These three tools store user's information in central space, and automatically fills in the fields with the saved credentials once the user revisit the page.

Some studies (*e.g.*,Winckler et al. [14] Bownik et al. [15] Wang et al. [16]) explore the use of semantic web for developing data binding schemas. The data binding schemas are essential techniques helping connect user interface elements with data objects of applications. This technology needs an ontology to perform the data integration. Instead of focusing on custom ontology, some binding schemas rely on the emergence of open standard data types, such as Microformats [17] and Micodata [18]. Winckler et al. [14] explore the effectiveness of the data schemas and the interaction techniques supporting the data exchange between personal information and web forms. Wang et al. [6] propose an intelligent framework to identify similar user input fields among different web applications by clustering them into different semantic groups. Araujo et. al [7] propose a concept-based approach, using the semantic of concepts behind the web form fields, for automatically filling out web forms. Some studies (*e.g.*, [19]) require apriori ([20]) tagging of websites, or a manually crafted list that includes the labels or names of input element to describe a semantic concept. These approaches

[13] `https://dl.dropboxusercontent.com/u/42298856/icwe2014.html`

can only be applicable to a specific domain or need explicit advice from the user. Hartman and Muhlhauser [4] present a novel mapping process for matching contextual data with UI element. Their method can deal with dynamic contextual information like calendar entries.

All the above approaches do not identify the variety of input parameters and treat input parameters equally by attempting to fill them into input parameters. They do not take into consideration the meaning of input parameters. In this study, we study the nature of input parameters and try to understand them from the end-user's point of view, not just by the similarity between the user inputs and the input parameters. Our study shows that taking into consideration that characteristics of input parameters via the identified categories significantly improves the performance of pre-filling.

7 Conclusion and Future Work

Reusing user inputs efficiently and accurately is critical to save end-users from repetitive data entry tasks. In this paper, we study the distinct characteristics of user input parameters through an empirical investigation. We propose four categories, *Volatile, Short-time, Persistent and Context-aware*, for input parameters. The proposed categories help pre-filling tool builders understand that which type of parameters should be pre-filled and which ones should not.

In this paper, we propose an ontology model to express the common parameters and the relations among them. In addition, we propose a WordNet-based approach to update the ontology to include the unseen parameters automatically. We also propose an approach to categorize user input parameters automatically. Our approach for category identification uses the proposed ontology model to carry the category information.

Through an empirical study, the results show that our proposed categories are effective to explain the unseen parameters. Our approach for updating ontology obtains a precision of 88% and a recall of 64% on updating the ontology to include unseen parameters on average. On average, our approach of category identification achieves a precision of 90.5% and a recall of 72.5% on the identification of a category for parameters on average. Moreover, the empirical results show that our categories can improve the precision of a baseline approach which does not distinguish different characteristics of parameters by 19%. Our proposed categories of input parameters can be the guidelines for pre-filling tool builders and our ontologies can be consumed by existing tools.

In the future, we plan to expand our definition of categories for input parameters by considering the scenarios of pre-filling web forms by multiple end-users. We plan to recruit end-users to conduct user case study on our approach.

Acknowledgments. We would like to thank all the IBM researchers at IBM Toronto Laboratory CAS research for their valuable feedback to this research. This research is partially supported by IBM Canada Centers for Advance Studies.

References

1. Rukzio, E., Noda, C., De Luca, A., Hamard, J., Coskun, F.: Automatic form filling on mobile devices. Pervasive Mobile Computing 4(2), 161–181 (2008)
2. Mozilla Firefox Add-on Autofill Forms, `https://addons.mozilla.org/en-US/firefox/addon/autofill-forms/?src=ss` (last accessed on February 4, 2014)
3. Google Chrome Autofill forms, `https://support.google.com/chrome/answer/142893?hl=en` (last accesed on February 4, 2014)
4. Hartman, M., Muhlhauser, M.: Context-Aware Form Filling for Web Applications. In: IEEE International Conference on Semantic Computing, ICSC 2009, pp. 221-228 (2009)
5. Toda, G., Cortez, E., Silva, A., Moura, E.: A Probabilistic Approach for Automatically Filling Form-Based Web Interfaces. In: The 37th International Conference on Very Large Data Base, Seattle, Washington, August 29-September 3 (2011)
6. Wang, S., Zou, Y., Upadhyaya, B., Ng, J.: An Intelligent Framework for Auto-filling Web Forms from Different Web Applications. In: 1st International Workshop on Personalized Web Tasking, Co-located with IEEE 20th ICWS, Santa Clara Marriott, California, USA, June 27 (2013)
7. Araujo, S., Gao, Q., Leonardi, E., Houben, G.-J.: Carbon: domain-independent automatic web form filling. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 292–306. Springer, Heidelberg (2010)
8. Xiao, H., Zou, Y., Tang, R., Ng, J., Nigul, L.: An Automatic Approach for Ontology-Driven Service Composition. In: Proc. IEEE International Conference on Service-Oriented Computing and Applications, Taipei, Taiwan, December 14-15 (2009)
9. WordNet, `http://wordnet.princeton.edu/` (last accessed on March 25, 2013)
10. Yin, R.K.: Case Study Research: Design and Methods, 3rd edn. SAGE Publications (2002)
11. RoboForm, `http://www.roboform.com/` (last accessed on March 25, 2013)
12. LastPass, `http://www.lastpass.com/` (last accessed on March 25, 2013)
13. 1Password, `https://agilebits.com/` (last accessed on March 25, 2013)
14. Winckler, M., Gaits, V., Vo, D., Firmenich, S., Rossi, G.: An Approach and Tool Support for Assisting Users to Fill-in Web Forms with Personal Information. In: Proceedings of the 29th ACM International Conference on Design of Communication, SIGDOC 2011, October 3-5, pp. 195–202 (2011)
15. Bownik, L., Gorka, W., Piasecki, A.: Assisted Form Filling. Engineering the Computer Science and IT, vol. 4. InTech (October 2009) ISBN 978-953-307-012-4
16. Wang, Y., Peng, T., Zuo, W., Li, R.: Automatic Filling Forms of Deep Web Entries Based on Ontology. In: Proceedings of the 2009, International Conference on Web Information Systems and Mining (WISM 2009), Washington, DC, USA, pp. 376–380 (2009)
17. Khare, R.: Microformats: The Next (Small) Thing on the Semantic Web? IEEE Internet Computing 10(1), 68–75 (2006)
18. Hickson, I.: HTML Microdata, `http://www.w3.org/TR/microdata` (last accessed on March 25, 2013)
19. Stylos, J., Myers, B.A., Faulring, A.: Citrine: providing intelligent copy-and-paste. In: Proceedings of UIST, pp. 185–188 (2004)
20. Apriori, `http://en.wikipedia.org/wiki/Apriori_algorithm` (last accessed on March 25, 2013)

Analysis and Evaluation of Web Application Performance Enhancement Techniques

Igor Jugo[1], Dragutin Kermek[2], and Ana Meštrović[1]

[1] Department of Informatics, University of Rijeka,
Radmile Matejčić 2, 51000 Rijeka, Croatia
{ijugo,amestrovic}@inf.uniri.hr
http://www.inf.uniri.hr

[2] Faculty of Organization and Informatics, University of Zagreb,
Pavlinska 2, 42000 Varazdin, Croatia
dkermek@foi.hr
http://www.foi.unizg.hr

Abstract. Performance is one of the key factors of web application success. Nowadays, users expect constant availability and immediate response following their actions. To meet those expectations, many new performance enhancement techniques have been created. We have identified almost twenty such techniques with various levels of implementation complexity. Each technique enhances one or more tiers of the application. Our goal was to measure the efficiency and effectiveness of such techniques when applied to finished products (we used three popular open source applications). We argue that it is possible to significantly enhance the performance of web applications by using even a small set of performance enhancement techniques. In this paper we analyse these techniques, describe our approach to testing and measuring their performance and present our results. Finally, we calculate the overall efficiency of each technique using weights given to each of the measured performance indicators, including the technique implementation time.

Keywords: Web application, performance, enhancement, techniques.

1 Introduction

Web applications (WAs) have become ubiquitous allowing anyone, even with only basic IT knowledge, to start an online business using a free and open sourced WA or a commercial one. There are three basic factors that have led to great importance of their performance today. First, the spread of broadband Internet connections has changed visitors expectations and tolerance to waiting for the application to respond, lowering the expected time to 1 or 2 seconds. The second factor is the increasing workload generated by the constantly growing number of Internet users. The third factor is the new usage paradigm (user content creation = write-intensive applications) that has been put forth by Web 2.0. All this has increased the pressure on performance of web applications. Our research had the following objectives: a) make a systematic overview of various techniques for enhancing WA performance, b) experimentally test, measure and evaluate the effectiveness of each technique, and c) measure the effect of these techniques on WA quality characteristics. The hypotheses we set were: a) it is possible to significantly increase application performance on the same hardware basis, independently of its category (type), by using even a small subset of performance enhancement techniques and b) implementation of performance enhancement techniques has a positive effect on the quality characteristics of such applications (efficiency, availability, reliability). The first hypothesis will be confirmed by achieving a 30% or more increase in throughput, while keeping the 90% of response times under 2 seconds and the average CPU usage under 70%. The second hypothesis will be confirmed by achieving a 10% or more decrease in average CPU usage, while still enhancing the throughput for at least 30%. In order to confirm our hypotheses experimentally, we selected three well known open source applications of different categories. The applications that we selected were: a) Joomla content management system (marked APP1 in this paper), b) PhpBB online community system (APP2) and c) OsCommerce e-commerce system (APP3). We have selected these applications because they have been under development for a long period of time; they are the solution of choice for many successful web companies and are used by millions of users every day.

The paper is organized into six sections. Section Two describes the theoretical foundation of this research and situates the work in the area. Section Three presents some motivations for performance enhancement of web applications, possible approaches and an overview of our analysis of performance enhancement techniques. In Section Four we present our experiment and discuss the results. In Section Five we calculate the effectiveness of each technique which suggest implications for the problem of performance enhancement technique selection in relation to the type and workload of WA whose performance we are trying to enhance. Finally, Section Six draws conclusions and suggests further work.

2 Background

There are three basic approaches to enhancing performance of WAs. While caching and prefetching are well known from other areas of computing, response time and size minimization is a relatively new approach developed by some of the most visited applications on the World Wide Web (Web) today. In this section we will point out some of the most important work within these approaches. Caching is one of the oldest methods of performance enhancement used in various information systems. In the area of web application development, caching has been analyzed in [18], [14], [5], [2] and [24], while various caching strategies have been analyzed in [25]. Another expanding area of performance enhancement is prefetching, i.e. preparing and/or sending data that is most likely to be requested after the last request. Domenech analyzed the performance of various prefetching algorithms in [8] and created prefetching algorithms with better prediction results in [7]. One of the crucial elements of prefetching is the prediction on which content will be requested next. Prediction is usually based on the analysis and modeling of user behavior as described [9], or by extracting usage patterns from web server logs, which has been done in [16] and in [11]. Adding a time component to better determine the optimal time for prefetching web objects was suggested in [15]. The latest approach is response time and size minimization. This approach is based on the experiences and methods developed by professionals that constructed some of the most popular applications on the Internet today, such as like Yahoo.com [26] and Flickr.com [10]. As said earlier Web 2.0 and AJAX have caused a paradigm shift in WA usage, which also brought on a change in the nature of WA workload. This was analyzed in [21],[20] and in [19]. Throughout 2009 and 2010, authors have studied the possibilities for the overall application performance enhancement [22], [6]. In this paper we analyze techniques based on all the mentioned approaches and measure their effect on the performance of WAs.

3 Performance of Web Applications

The quality, and with that, the performance of WAs is primarily defined by the quality of its architecture. Badly planned elements of application architecture can limit or completely block (by becoming a bottleneck) the expected or needed levels of performance. With the new focus on the importance of application performance, production teams have begun to implement performance risk management in all phases of its lifecycle. According to the Aberdeen group research [1], companies that have adopted performance management in the form of the "Pressures, Actions, Capabilities, Enablers" (PACE) model have seen a 106% increase in availability, 11.4 times better response times and 85% problem solving rate before the problem becomes obvious to their users. However, there are already many WAs used worldwide, and with the rise in the number of users, their performance has to be enhanced to meet the desired service levels. The motivation for performance enhancement usually comes from three main directions: a) the desire to increase the number of users or profit, b) expected and planned peak workload periods and c) performance enhancement based on a business plan, or inspired by a real or expected (projected) performance problem with a goal of ensuring availability and efficiency under increased workload. Performance enhancement is always a tradeoff between the number of opposing factors like hardware and working hours investment limits, desired response times and throughput values, etc. In our research we had to balance between two limiting factors (response time and CPU usage) while trying to maximize a third one (throughput).

These factors can be visualized as given in Figure 1. The X axis shows the response time, with a maximum value of eight seconds, which was considered by some authors to be the limit of tolerable waiting time a few years ago. We believe that today, this limit for an average user is much lower so we have set the average response time for 90% of requests to 2 seconds. The Z axis displays the average CPU usage during load tests. The usage limit here has been set to 70% according to some best practices in the hardware industry (constant usage over 70%

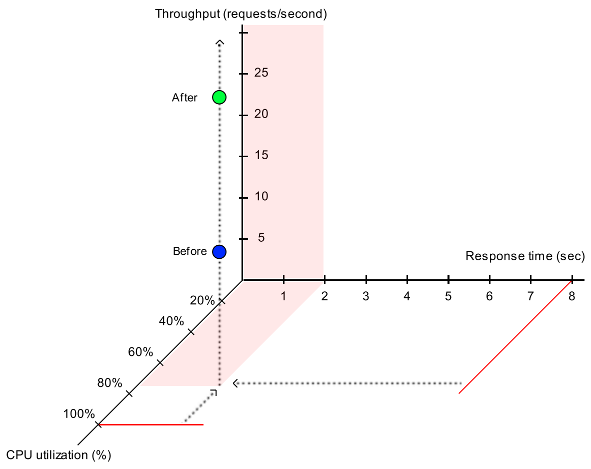


Fig. 1. Performance enhancement area

significantly raises malfunction probability and shortens the Mean Time Between Failure (MTBF), along with the Total Cost of Ownership (TCO)). In this way, we have set the margins of an area within which we can try to achieve the highest possible throughput (Y axis) by implementing various performance enhancement techniques.

Performance enhancement requires a complete team of experts from all fields of WA development - from front-end (HTML, CSS, JavaScript) to back-end (database design, SQL queries, DBMS configuration and administration) in order to plan and implement performance enhancement techniques, as well as verify changes through benchmarking. Such teams must also check the performance of application architecture at each level, as some techniques may have hidden effects that can propagate throughout the application and even cause a decrease in the performance at another level. Some techniques require additional services that must be installed on the server. This increases the complexity of application architecture and increases CPU and memory usage (if additional services are running on the same server), which has to be taken into account when doing post-implementation load testing. When trying to enhance the performance of a WA, a starting approach can be general systems theory that considers the application to be a black box with a large number of requests per second as input, and a large number of responses per second as output. When we start to decompose the black box into subsystems, we identify the ones that spend most of the time needed to generate responses. If one subsystem takes 80% of this time, then our choice is to try to enhance the performance of that subsystem, instead of another subsystem that spends just 5% of the time. After enhancing performance of one subsystem, another becomes the primary bottleneck. It is an iterative process of defining quantitative parameters that define the acceptable behavior of the system, benchmarking, identifying of bottlenecks, modifying, and benchmarking again. Performance measuring is a process of developing measurable indicators that can be systematically tracked for supervising advances in service level goal fulfillment. With WAs, there is usually a discrepancy between the required or expected, and real performance results. Performance levels can address quality, quantity, time or service price. There are two basic approaches to performance measuring: benchmarking and profiling. While benchmarking answers the question how well the application works, profiling answers the question why does the application have such performance. In this research we have used both approaches to measure the overall performance, as well as to profile the response time structure of individual requests. There are dozens of parameters that can be measured while benchmarking a production or a staging version of an application, which are selected based on identified bottleneck or performance enhancement goals. It is usually not possible to test the application in production environment, so we use test versions, or test part(s) of a cluster, and use an artificial workload. Workload modeling is a very important part of the performance benchmarking process. To achieve the correct measurements, the artificial workload must mirror the sample of the real workload, i.e., live users of the application. This can be achieved using different data sources: real time user observations, planned interactions or log file analysis. Test workload implementation must be as realistic as possible; otherwise, incorrect data may be collected and wrong decisions could be made.

Peformance Enhancement Techniques. Over the years of Web development and research, many different performance enhancement techniques (some more, some less complex to implement) have been proposed and used in practice. As the nature and complexity of the content offered online have changed [23], so have the techniques for performance enhancement. During our preliminary research of this subject we have identified many different techniques aimed at enhancing the performance of multi-tier web application architecture. Some of them are now obsolete; some are effective only with highly distributed systems; others are effective only at most extreme workloads where even 1kB makes a difference; etc. In our research we have analyzed those techniques that are currently most widely used and can be implemented on most WAs in use today - 19 techniques in total (as shown in Table 1). These techniques have been analyzed in the following manner: we defined the objective of each technique and the way in which it tries to achieve its objective, described the implementation process and gave an estimate of duration. Selected techniques have been sorted by the tier of WA architecture that they affect. As expected, the lowest number of techniques affect the business logic tier of application architecture. Although the business logic of an application can cause the bottleneck effect, it is by far the most expensive and complex problem to resolve which cannot be done by any enhancement technique. If the business logic of the application appears to be the causing the bottleneck, that is an indication of bad architecture decisions or implementation. When functioning normally, code execution takes from 5-15% of total response

Table 1. List of analyzed techniques

	No code changes required		Code changes required
T1	Caching objects using HTTP expires	T2	Reducing number of HTTP requests
	Compressing textual content		Positioning of HTML content objects
	JavaScript minification and obfuscation		Reducing number of DNS requests
	Reducing size of graphical objects		Managing redirects
	RDBMS conguration tuning		Delayed content download
T3	Caching of interpreted PHP scripts		Enhancing performance of core logic
	Standalone PHP application server	T4	Caching query results (Memcache)
	Choice of web server		Query and database optimization
T5	Web server conguration tuning		Using stored procedures
T6	Caching pages using a proxy server		

(round trip) time and as such it is not a particularly fruitful subsystem for performance enhancement. According to [26], most of the response time is spent on content delivery (up to 80%) and query execution (database lag), so most of the analyzed techniques try to enhance the performance of these subsystems.

All the techniques displayed in Table 1 have been analyzed in detail. Due to the limited scope of this paper, we omit a detailed analysis which can be found in [13]. The analysis consists of 6 properties: 1) Tier - of the WA architecture affected by the technique, 2) Idea - basic explanation of how the technique increases performance, 3) Implementation - how is the technique implemented, 4) Time - how long it took us (on average if not stated differently) to implement this technique in the three applications used in this research. Implementation time for these techniques will vary with respect to the size and complexity of the application (e.g. amount of code, number of servers), 5) Expected results - performance characteristics the technique affects and 6) Verification - how is the performance gain measured. For the experimental part of our research we have selected six techniques with various degrees of complexity and implementation time. Those techniques have been selected based on the assumption that they will have the highest impact on the overall performance of WAs that have been used in this research. A list of analyzed techniques, as well as those that have been selected for experimental testing, can be seen in Table 1. Techniques selected for the experimental part of our research are labeled with T1-T6.

4 Performance Testing and Analysis

In the experimental part of our research we used three open-source WAs of different types/categories: portal (Joomla), community (PhpBB), and e-commerce (OsCommerce) to implement and test the effectiveness of performance enhancement techniques.

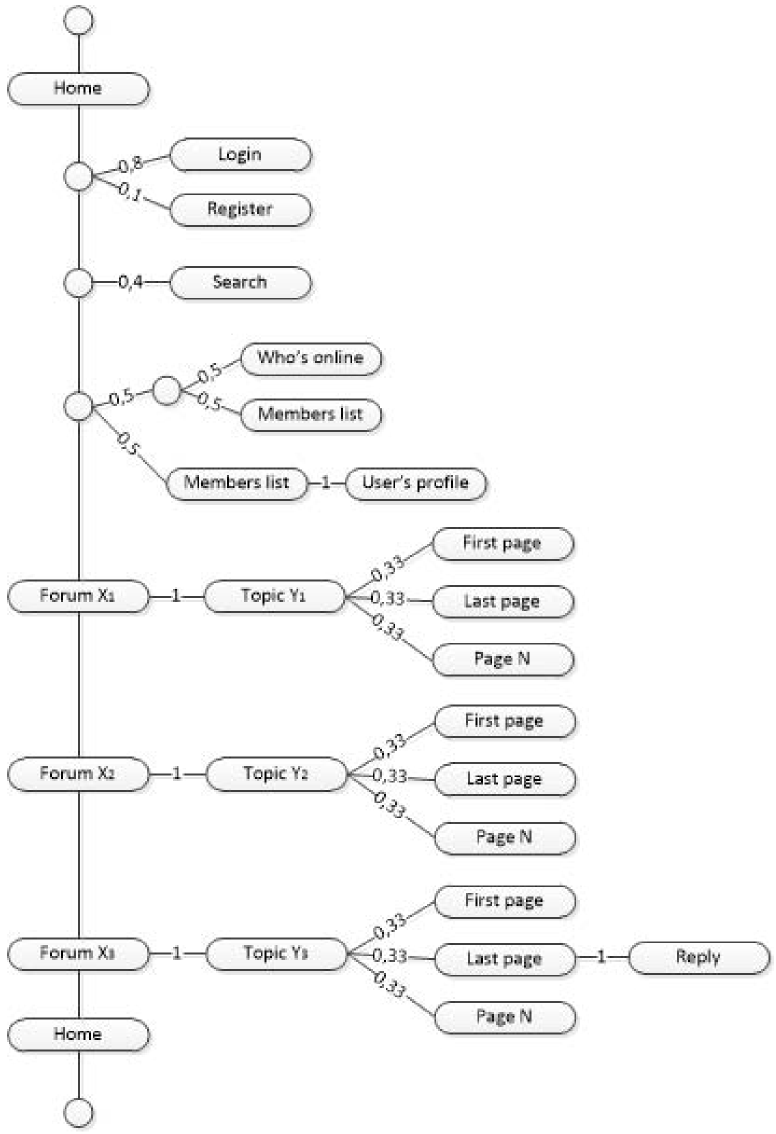


Fig. 2. Workload model for APP2 (Forum)

4.1 Preparation Phase

The first step of the research was to develop a realistic workload model (for all three applications) that will be used in all performance tests. Test workloads were modeled using various approaches: server log analysis (requires shell scripting to extract usage patterns), planned interactions (set by probability or importance, e.g. product browsing in an e-commerce application has the highest probability) and real user auditing. More about these approaches can be found in [17]. Figure 2 displays the workload model developed for load testing APP2. All actions (HTTP requests) on the baseline will be executed each time the test is performed, while others have a probability indicated on the connecting lines. The test can be described in the following way: all users start at the home page, most of them log on to the system, while some register and some (10%) will remain as guests. About 40% of users will use the search option. Visitors then look at one of the three popular pages (members list, who's online and a users profile). Then they look at three forums (two of them are most popular while the third one is randomly selected) and look at one topic inside those forums. When accessing the topic page some users start at the first post, others go directly to the last page and some select one of the pages of the topic. A third of the visitors will post a message to the forum. Finally, they will go back to the home page (check that there are no more new posts) and leave the page. Workload models for other applications were developed in a similar fashion (ie. most popular articles on the portal, featured products in the webshop). To implement workload models and perform load tests the JMeter [3] tool was used. With complex test models we simulated many requests from the individual user which were randomized by timers (simulating pauses for reading, thinking and form submissions) and random branching (between sets of actions). To ensure randomization of URL requests, test data sources (TXT files) were prepared from the real WA databases. These TXT files are constructed by selecting a subset of primary key identifiers from the real application database and written in a tab delimited format. JMeter then reads one line from the TXT file for test run and adds identifiers to the generated HTTP requests (links). In this way we can simulate the behavior of real users (reading about products, commenting

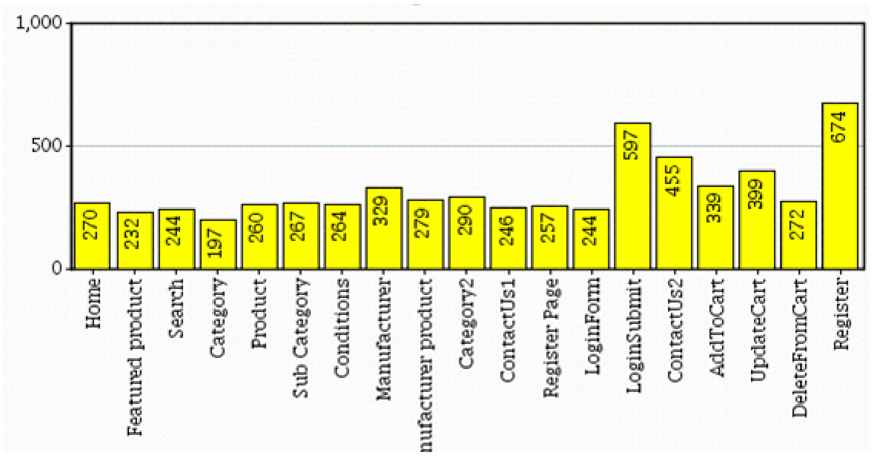


Fig. 3. JMeter aggregate graph

products, adding products to cart, making purchases, etc.). After the workload models were designed and implemented we tested them and verified that they executed correctly (reading identifiers from TXT files, adding them to HTTP requests, all response codes 200, database inserts correct, etc.). After the workload models were developed and implemented as load tests in JMeter we had to find a way to log data about hardware utilization during the tests. We decided to use Linux command line tool **vmstat** which proved to provide enough data on a fine enough time scale with a small enough footprint on server CPU usage. The next step was preparing data visualizations for the analysis phase. The first visualization was done by JMeter. After performing a test, JMeter displays data about each performed request, such as response time and server response, and calculates throughput and average response times. JMeter then offers to generate various visualizations based on this data. In relation to our first measuring constraint (90% of all requests under 2 seconds) we chose to generate an aggregate graph (shown in Figure 3) that displays average response times for each HTTP request (link) defined in the workload model. We have also used **Httpload** [12] and **ab** [4] for fast tests and to verify the results recorded by JMeter. The second visualization was done after the collected hardware utilization data was processed using an **awk** script (to remove unnecessary columns and calculate column averages) and handed over to a **gnuplot** script we developed for this research. Figure 4 demonstrates the output of our gnuplot script. Duration of the test is shown on the X axis in each chart. This visualization

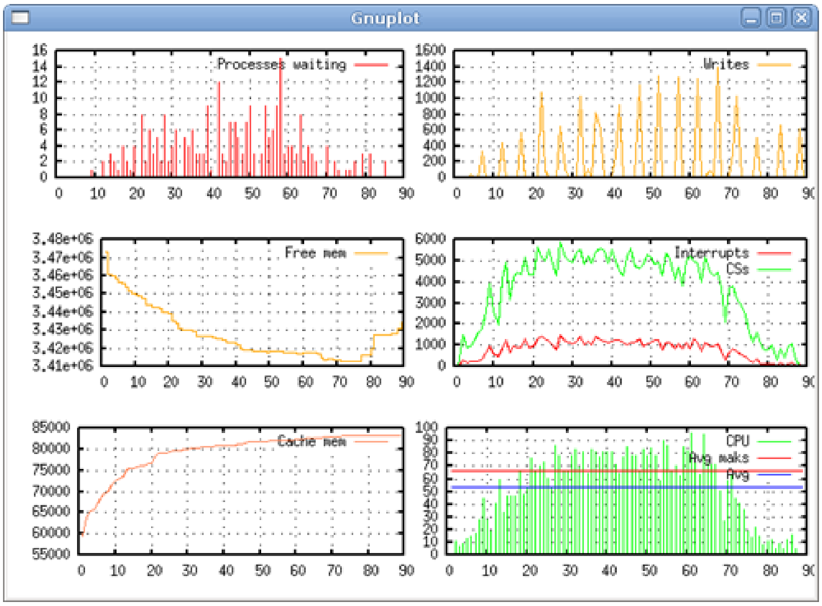


Fig. 4. Hardware utilization visualization (vmstat >awk >gnuplot)

consists of six charts representing various hardware utilization metrics received from vmstat. Some charts contain two data series. The top most graph on the left hand side displays the number of processes waiting during the test (lower number of processes waiting means lower response time), middle-left displays the amount of free memory, and bottom-left the amount of memory used for cache (more data in the cache means lower response time and higher throughput). The top right hand side graph displays the number of writes on the disc (the more files are cached, the less time is spent fetching data from the hard drive, resulting in lower response time, higher throughput and lower CPU usage). The middle-right graph contains two data sets: the number of interrupts and the number of context switches (the higher these values are, the higher CPU usage is, which results in higher response time and lower throughput). The bottom-right graph displays CPU usage during the test and 2 lines: one is the 70% limit set as one of the two main constraints, and the other is the calculated average of CPU usage during the test. The calculated average for the duration of the test had to be less than or equal to 70%. We increased the load on the test server by increasing the number of simulated users in JMeter test plans, until we came as close as possible to one or both limiting factors. Then we performed the three official tests (following the procedure described the following section). All tests were performed in an isolated LAN consisting of a load generator PC, a 100Mbit passive switch and the testbed PC. The testbed PC had the following hardware configuration: Asus IntelG43 Chipset, CPU: Intel Core2Duo 2.5Ghz, 1MB Cache, RAM 4GB DDR2, HDD WD 250GB SATA, OS Debian Linux. Network bandwidth was not a bottleneck in any of the preparation tests so we did not measure network usage.

4.2 Testing Phase

Before starting load tests we analyzed each application (file structure, number of files included per request, number of graphical objects, measured individual request response time). An important measure was the number of SQL queries performed per request. While the first two applications performed 28 and 27 queries, the third one did 83, which was a probable bottleneck. The testing phase consisted of 3 stages. First, we measured the initial performance of all three applications (after installation, using a test data set). Second, we implemented one of the selected performance enhancement techniques and repeated the measurement to determine the effect of the technique on performance (we performed this test 3 times). Then the application was restored to its initial state. This process was repeated for each technique. Third, we implemented all of the selected techniques to all three WAs and made the final performance measurement to determine final performance enhancement achieved using these techniques. In total we performed over 200 load tests. When doing performance testing, ensuring equal conditions in order to achieve the correctness of the acquired data is obligatory and was integrated in our testing procedure:

```
For APP1, APP2, APP3
  For Technique T1 - Tn
    Implement technique Tn
    Run multiple tests until as close as possible to two
    limiting factors // increment number of simulated users
    Reset()
    Restart server (clear memory, cache, etc.)
    Run a warm-up test (prime caches)
    Repeat 3 times
      Run test and gather data (JMeter, vmstat)
      Reset()
    Remove implementation of technique Tn

Reset()
  Reset database to initial state (remove inserts from the
  previous test)
  Reset server logs
```

We have performed each test three times for each implemented technique in order to get a more robust measurement and reduce the possibility of errors. Average values have been calculated from these measurements and are displayed in Table 2. Before starting our measurements, we set the throughput as the key performance indicator. Our goal was to enhance it using various performance enhancement techniques. **The limiting factors were: a) average response time for the 90% of requests had to be less than 2 seconds and b) average CPU utilization had to be kept below 70%**. First, we present the results obtained from the initial testing, then the results for each technique, and, lastly, the final testing results. These results display only the values of the aforementioned key performance characteristics. During each benchmark we collected data for 10 performance indicators which will be taken into consideration later. Second, we display the overall results that confirm our hypotheses. Third, we take into consideration all the data obtained from benchmarking and add weights to each performance indicator in order to define the effectiveness of each technique.

4.3 Results

The summary of our measurements is presented in Table 2. Let us first explain the structure of the Table. The markings in the first column are: B (for Beginning) - this row displays data about the initial performance of un-altered applications (after installation); T1-T6 (for Technique) - these rows display data about performance after each of the techniques was implemented individually; F (for Final) displays data about the performance of applications after all six techniques were implemented and finally C (for Comparison) displays data about the performance of applications under the same workload used in the B row. For each application there are three columns: The Response time (in miliseconds) is given

Table 2. Overall load testing results

-	APP1 Portal			APP2 Forum			APP3 Webshop		
	RT (ms)	T (R/s)	U (%)	RT (ms)	T (R/s)	U (%)	RT (ms)	T (R/s)	U (%)
B	834	8,3	69	791	15,2	66	794	24	66
T1	1025	8,3	68	661	**16,8**	67	794	24	66
T2	816	8,4	67	789	16,4	67	857	**27**	67
T3	793	**14,4**	68	815	**32,3**	69	1218	**35**	67
T4	858	**13,5**	68	958	**16,8**	69	993	25	67
T5	823	8,6	67	527	15,8	66	726	25	69
T6	958	8,4	69	954	17	70	1670	25	66
F	614	24,9	68	671	38,5	68	1740	35	64
C	91	8,5	21	58	16	23	442	24,1	48

LEGEND: RT= Response time, T = Throughput, U = Utilization, ms = miliseconds, R/s = Requests per second, CPU = average percent of CPU utilisation (measured by vmstat).

in the first column, the Throughput (in requests/second) in the second, and the CPU utilization (average, in %) in the third column. Although some techniques appear to increase the average response time and do not increase the overall throughput, they have a positive effect on other performance indicators that were measured and will be discussed later on. Furthermore, the first technique reduces individual response time which is not visible here, but can be observed in individual response profiling using YSlow (a Firefox profiling add-on). It can be seen that PHP script caching using APC (T3) clearly has the biggest impact on the performance of all tested applications. Storing SQL query data in memory cache using Memcache (T4) follows closely being the second most effective with two of three applications. We can also see that different techniques have different effects on each application. The comparison row (C) demonstrates the performance of applications under the same beginning workload, which was the highest possible before the implementation of all performance enhancement techniques. This data shows how the application would perform under normal daily workload, after its performance has been enhanced using the 6 mentioned techniques. In this row we can see that the throughput value has decreased significantly. This is caused by the fact that JMeter uses the number of simulated users/second (which is one of the arguments used when starting the load test) to calculate throughput. To make the comparison we had to use the load that was used to make the initial performance measurement for the WAs in their original (un-enhanced) state. Therefore, the number of simulated users/second was much smaller than it was for the final versions of WAs with all 6 techniques implemented. The result of final load testing also demonstrates that combining all of the techniques has a cumulative effect, because the maximum throughput for APP1 and APP2 is higher than the contribution of any individual technique. This was expected based on the effect of performance enhancement techniques

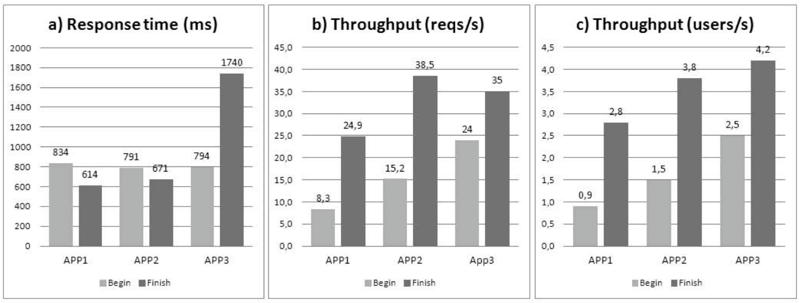


Fig. 5. Final performance enhancement results

on other performance indicators we observed (which will be described in the following section). This was not the case with APP3, whose maximum throughput was limited by database lag (due to the large number of database queries per request (an architectural problem)). To visualize the performance improvements we chose three key indicators - response time, throughput measured by number of requests/second (displayed in Column "T" in Table 2) and throughput measured by the number of users/second (not displayed in Table 2, but measured along with other indicators listed in Table 3). The differences between the values of these indicators in the first(B) and final (F) measurements are given in Figure 5 (a), (b) and c) respectively. The average response time for APP1 and APP2 was reduced even with a much higher workload while the throughput was increased by almost three times. For APP3, our hypothesis was confirmed (we achieved a 30% increase in throughput) but due to the very large number of database queries performed for each request, the increase in throughput was almost 2 times smaller than with APP1 and APP2. The increase in response time is caused by the same problem but within the limit of 2 seconds average for 90% of requests.

5 Calculating Technique Effectiveness

Although the results obtained show that it is possible to significantly enhance the performance of various types of WAs by using just a small subset of performance enhancement techniques, we were interested in defining an overall "quality indicator" of used techniques for each application type, and checking whether they appear in the same order with all WAs. This would mean that there is a uniform top list of techniques that can be implemented on any web application. To precisely determine the efficiency and effectiveness of a technique, we took all the recorded indicators into consideration. The full list of indicators whose values have been recorded during testing is displayed in Table 3. The most important indicator (after the three previously mentioned) is the time needed to implement the technique, which we recorded for each technique. Each indicator is given a

Table 3. List of indicators with appointed weights

No	Name	Weight	Proportional	Acquired from
1	Response time	5	Inversely	JMeter
2	Throughput (requests/sec)	5	Directly	JMeter
3	Throughput (users/sec)	5	Directly	JMeter
4	Utilization (cpu)	2	Inversely	vmstat
5	Processes waiting	4	Inversely	vmstat
6	RAM usage	2	Directly	vmstat
7	Cache mem. usage	3	Directly	vmstat
8	Disc usage (writes)	3	Inversely	vmstat
9	Context switches	4	Inversely	vmstat
10	Implementation time	5	Inversely	Measured / Estimated

weight (range 1-5), marking its importance in the overall performance gains. The weights were given according to our perception of each indicators importance in the overall performance enhancement. The column titled "Proportional" indicates whether the measured indicator is directly or inversely proportional to the technique effectiveness. Directly proportional means the higher the measured value, the better, while inversely proportional means the lower the value, the better, e.g. an increase of requests per second is directly proportional while the implementation time is inversely proportional. We measured the time needed for one developer to implement the technique (change server configuration, change application configuration, change code of the application) in each application. To determine the "quality indicator" of a technique we used the following procedure and equations.

```
For each WA (APP1, APP2, APP3)
 For each technique (T1-T6)
    For each of 10 performance indicators
       Calculate the effect E of indicator n using the value of
  indicator Vn in relation to the indicators maximum value change
 (Cnmax) and minimum indicators value change (Cnmin). Depending on
  whether the indicator is inversely proportional or directly
  proportional formulas (1) or (2) are used (respectively)
```

$$E_n = C_{n_{max}} - V_n / C_{n_{max}} - C_{n_{min}} \ . \tag{1}$$

$$E_n = V_n - C_{n_{min}} / C_{n_{max}} - C_{n_{min}} \ . \tag{2}$$

```
Calculate indicator weight using (3)
```

$$W_i = W_n / W_{max} \ . \tag{3}$$

`Sum up to get the technique efficiency using (4)`

$$\sum T_i = \sum_{i=1}^{10} W_i * E_i \ . \tag{4}$$

In this way, we calculated the top list of performance enhancement techniques with respect to type/category of the WA. The results are displayed in Table 4. Technique 1 (Reducing number of HTTP requests) was not implemented on APP 3 (marked "N/A" in Table 4.) because there were not enough graphical objects that could have been merged into a single larger one. It is clear that the order (ranking) of techniques is different for each type of application. A few important conclusions can be made from these calculations and will be used as problem guidelines in our future work. First, we don't have a framework that defines which techniques to use for each type of WA. It is clear that the subset of techniques to be used for performance enhancement must be tailored to the specific application. Secondly, there are a number of factors that influence the decisions about the techniques to be used such as: the goals of performance enhancement (what aspect of performance are we trying to enhance), the type of content the WA delivers (e.g. text, graphic objects, large files, video, etc.) and the specific workload. In our future work we will repeat these measurements on a larger number of (various types of) WAs and try to develop and verify such a framework for identifying a subset of techniques that yields the best results based on these factors.

Table 4. Overall ranking of performance enhancement techniques effectiveness

No	Technque	APP1	APP2	APP3
1	Reducing number of HTTP requests	5	3	N/A
2	Caching objects using HTTP Expires	3	1	3
3	Caching of interpreted PHP scripts (APC)	2	2	1
4	Caching query results (memcache)	1	6	4
5	Web server conguration tuning	6	5	4
6	Caching objects using proxy server (Squid)	4	6	2

6 Conclusion

Static websites are rapidly being replaced by web applications, and the ever increasing number of Internet users demands more and more functionality while expecting lower and lower response time. Web 2.0 has brought about a paradigm shift which changed the structure of workload, moving it from read-intensive to write-intensive. Therefore, the performance of WAs has become one of the focal points of interest of both scientists and professionals in this field. The goal of performance enhancement has to be set before any of the techniques are implemented or tests performed. This goal depends on the problem perceived in the performance of an application and can be aimed at any aspect of its performance (e.g. minimizing CPU or memory usage). In this research, our goal was to maximize throughput and lower response time of different finished systems (web applications) on the same hardware basis. Performance measurement itself is a complex process that requires careful monitoring of the real workload, identification of bottlenecks, planning and modelling test workloads, identifying key characteristics, goals, technical knowledge on all elements of the content creation and delivery system, etc. We have proved that it is possible, in a controlled environment at least, to significantly enhance the performance of WAs using just a small set of performance enhancement techniques with a total implementation time ranging from 10 to 50 working hours for applications running on one multiple-role (e.g. web, proxy, application) server. We found that the results of each technique vary from application to application and that further research is needed to develop a generalised framework that would take into consideration all the factors mentioned above (goals, content type, system architecture, etc.) and suggest what techniques would be best suitable for a selected application.

References

1. Aberdeen Group: Application Performance Management, `http://www.aberdeen.com/Aberdeen-Library/5807/RA-application-performance-management.aspx`
2. Amza, C., Soundararajan, G., Cecchet, E.: Transparent Caching with strong consistency in dynamic content web sites. ICS Boston (2005)
3. Apache JMeter, `http://jakarta.apache.org/jmeter`
4. Apache Benchmark Tool, `http://httpd.apache.org/docs/2.0/programs/ab.html`
5. Bahn, H.: Web cache management based on the expected cost of web objects. Information and Software Technology 47, 609–621 (2005)
6. Bogardi-Meszoly, A., Levendovszky, T.: A novel algorithm for performance prediction of web-based software system. Performance Evaluation 68, 45–57 (2011)
7. Domenech, J., Pont, A., Sahuquillo, J., Gil, J.A.: A user-focused evaluation of web prefetching algorithms. Journal of Computer Communications 30, 2213–2224 (2007)
8. Domenech, J., Pont, A., Sahuquillo, J., Gil, J.A.: Web prefetching performance metrics: a survey. Performance Evaluation 63, 988–1004 (2006)
9. Georgakis, H.: User behavior modeling and content based speculative web page prefetching. Data and Knowledge Engineering 59, 770–788 (2006)
10. Henderson, C.: Building Scalable Web Sites. OReilly, Sebastopol (2006)
11. Huang, Y., Hsu, J.: Mining web logs to improve hit ratios of prefetching and caching. Knowledge-Based Systems 21, 149–169 (2008)
12. Http Load Tool, `http://www.acme.com/software/httpload/`
13. Jugo, I.: Analysis and evaluation of techniques for web application performance enhancement, Master of Science Thesis, in Croatian (2010)
14. Khayari, R.: Design and evaluation of web proxies by leveraging self- similarity of web traffic. Computer Networks 50, 1952–1973 (2006)
15. Lam, K., Ngan, C.: Temporal prefetching of dynamic web pages. Information Systems 31, 149–169 (2006)

16. Liu, H., Keelj, V.: Combined mining of Web server logs and web contents for classifying user navigation patterns and predicting users future requests. Data and Knowledge Engineering 61, 304–330 (2007)
17. Meier, J.D., Farre, C., Banside, P., Barber, S., Rea, D.: Performance Testing Guidance for Web Applications. Microsoft Press, Redmond (2007)
18. Na, Y.J., Leem, C.S., Ko, I.S.: ACASH: an adaptive web caching method based on the heterogenity of web object and reference characteristics. Information Sciences 176, 1695–1711 (2006)
19. Nagpurkar, P., et al.: Workload characterization of selected J2EE-based Web 2.0 applications. In: 4th International Symposium on Workload Characterization, pp. 109–118. IEEE Press, Seattle (2008)
20. Ohara, M., Nagpurkar, P., Ueda, Y., Ishizaki, K.: The Data-centricity of Web 2.0 Workloads and its impact on server performance. In: IEEE International Symposium on Performance Analysis of Systems and Software, pp. 133–142. IEEE Press, Bostin (2009)
21. Pea-Ortiz, R., Sahuquillo, J., Pont, A., Gil, J.A.: Dweb model: Representing Web 2.0 dynamism. Computer Communications 32, 1118–1128 (2009)
22. Ravi, J., Yu, Z., Shi, W.: A survey on dynamic Web content generation and delivery techniques. Network and Computer Applications 32, 943–960 (2009)
23. Sadre, R., Haverkort, B.R.: Changes in the web from 2000 to 2007. In: De Turck, F., Kellerer, W., Kormentzas, G. (eds.) DSOM 2008. LNCS, vol. 5273, pp. 136–148. Springer, Heidelberg (2008)
24. Sajeev, G., Sebastian, M.: Analyzing the Long Range Dependence and Object Popularity in Evaluating the Performance of Web Caching. Information Technology and Web Engineering 4(3), 25–37 (2009)
25. Sivasubramanian, S., Pierre, G., van Steen, M., Alonso, G.: Analysis of Caching and Replication Strategies for Web Applications. Internet Computing 11(1), 60–66 (2007)
26. Souders, S.: High Performance Web Sites. O'Reilly, Sebastopol (2007)

CRAWL·E: Distributed Skill Endorsements in Expert Finding

Sebastian Heil, Stefan Wild, and Martin Gaedke

Technische Universität Chemnitz, Germany
`firstname.lastname@informatik.tu-chemnitz.de`

Abstract. Finding suitable workers for specific functions largely relies on human assessment. In web-scale environments this assessment exceeds human capability. Thus we introduced the CRAWL approach for Adaptive Case Management (ACM) in previous work. For finding experts in distributed social networks, CRAWL leverages various Web technologies. It supports knowledge workers in handling collaborative, emergent and unpredictable types of work. To recommend eligible workers, CRAWL utilizes Linked Open Data, enriched WebID-based user profiles and information gathered from ACM case descriptions. By matching case requirements against profiles, it retrieves a ranked list of contributors. Yet it only takes statements people made about themselves into account. We propose the CRAWL·E approach to exploit the knowledge of people *about* people available within social networks. We demonstrate the recommendation process for by prototypical implementation using a WebID-based distributed social network.

Keywords: Endorsements, Expert Finding, Linked Data, ACM, WebID.

1 Introduction

Knowledge work constitutes an ever increasing share of today's work. The nature of this type of work is collaborative, emergent, unpredictable and goal-oriented. It relies on knowledge and experience [17]. Traditional process-oriented Business Process Management (BPM) is not well applicable to areas with a high degree of knowledge work [21]. Addressing this issue, non-workflow approaches [5], in particular Adaptive Case Management (ACM), gain more relevance [8].

ACM systems assist knowledge workers. They provide the infrastructure to handle dynamic processes in a goal-oriented way. Traditional BPM solutions feature a-priori processes modeling. Contrary to them, ACM systems enable adaptivity to unpredictable conditions. Uniting modeling and execution phases contributes accomplishing this adaptivity. A case represents an instance of an unpredictable process and aggregates all relevant data. For adapting it to emergent processes, case owners can add ad-hoc goals. There are cases where persons currently involved cannot achieve all goals. In these cases it is necessary to identify suitable experts based on the skills and experience required for that particular part of work.

The nature of knowledge work often implies cross-enterprise collaboration. It necessitates access to information about the persons involved, e.g., CV and contact data. It is unlikely that all potential collaborators use the same social network platform for storing personal information. Cross-platform relationships are hard to follow. Such "walled gardens" [1] would complicate expert finding. Distributed social networks are well-suited for the knowledge work domain. Companies or knowledge workers can host their own profiles. The profile can include work experience and skill information. Interlinking these distributed profiles establishes a social network. Such social network could overcome the data silo characteristic of walled gardens. This would enable crawling the network to identify experts.

Finding suitable workers for specific functions largely relies on human assessment. Assessors have to make their decisions depending on the requirements at hand. This decision making requires knowledge of potential contributors and their experience. The selection complexity increases with the amount of eligible contributors and work requirements. Human assignment does not scale well, especially not with web-scale processes [12]. Often work is assigned to workers who are not the most suitable experts available. This can cause mediocre outcomes and longer times to completion. Dealing with this problem requires software support for finding and addressing knowledge workers to contribute to cases.

In [15] we introduced CRAWL, an approach for Collaborative Adaptive Case Management. It leverages various Web technologies to automatically identify experts for contributing to an ACM case. CRAWL recommends a set of eligible workers. It uses Linked Open Data, enriched WebID-based user profiles and information gathered from project or case descriptions. We created a vocabulary to express the skills available and the skills required. It extends user profiles in WebID-based distributed social networks and case descriptions. CRAWL's semantic recommendation method retrieves a ranked list of suitable contributors whose worker profiles match the case requirements.

Problem. The skill information about a person is limited to the expressive power and will of this particular person. As a consequence, CRAWL only takes statements people made about themselves into account. Such statements, however, might be unspecific, exaggerated or even wrong. This affects the expert finding process and makes the assessor's task more difficult and time-consuming.

There are three possible kinds of statements about skills, as shown in Figure 1. The most basic form are *skill self-claims*, statements by someone claiming that he himself has a certain skill. *Skill assignments*, on the other hand, are statements by someone claiming that someone else has a certain skill. Statement claimed by someone for himself and confirmed by someone else are *skill affirmations.* We refer to these three kinds of skill statements together as *skill endorsements.*

With knowledge work increasingly becoming an important and widespread part of work [9] and ACM evolving as an approach addressing this type of work, we are convinced that enabling knowledge workers to find the right collaborators to contribute to multi-disciplinary cases impacts the performance of future enterprises [5]. The value add by skill endorsements will trigger a demand to incorporate them into distributed worker profiles and expert finding algorithms.

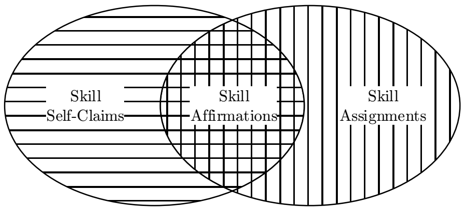


Fig. 1. Three kinds of skill endorsements

Overall Objective. To exploit the knowledge of people *about* people available within social networks, we aim at integrating skill assignments and skill affirmations in addition to skill self-claims into the distributed expert finding process.

Contributions. To contribute to the overall objective, we must achieve the following objectives to fully use skill endorsements in distributed social networks:

1. To enable expert finding in distributed social networks
2. To increase credibility of skill self-claims
3. To allow assigning skills the endorsee did not consider
4. To prevent unwanted skill endorsements
5. To express skill endorsements in distributed user profiles
6. To incorporate skill endorsements in distributed expert finding
7. To facilitate a differentiated consideration of skill endorsements

The paper is organized as follows: Section 2 illustrates the necessary objectives in order to achieve the overall objective. The background is provided in Section 3. We present the CRAWL·E approach in Section 4. Section 5 evaluates the approach. We discuss work related to ours in Section 6 and conclude the paper in Section 7.

2 Objectives of Distributed Expert Finding

This section describes the objectives in greater details. To illustrate the need for achieving each objective, we use different personae. All of them are knowledge-workers and members of a distributed social network. They have a different character and pursue different goals. Figure 2 shows the corresponding social graph. Black solid arrows indicate knows-relationships, blue dotted arrows symbolize endorsements. The personae are characterized in the following:

Alice wants to record her skills. She likes to include all skills from her current job, past jobs and education. Alice intends to record them in a way others can easily access them. She does not want to spend too much effort in achieving this goal.

Bob is a co-worker of Alice. He knows Alice very well because he worked together with her in many projects. Bob trusts Alice and Alice trusts him.

Casey is a case owner who wants to find and recruit the best persons for a job.

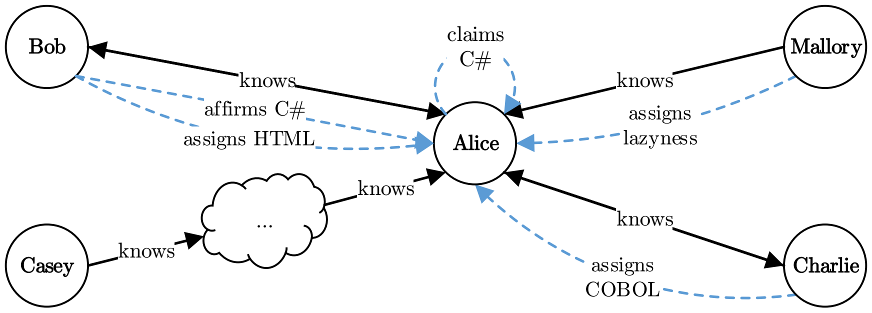


Fig. 2. Social Graph with Endorsements

Charlie is another co-worker of Alice. Compared to Bob, he is not that close to Alice. Charlie worked together with Alice in only one project long time ago.

Mallory is a bad guy. He dislikes Alice and wants to damage Alice's reputation.

Having described the personae that are used throughout this paper, we continue with outlining the objectives.

Objective 1: To enable expert finding in distributed social networks. Interoperability, compatibility, and portability of skill information is no issue in conventional centralized social networks like LinkedIn. Expert finding benefits from the (virtually) monolithic data layer of such social networks. By contrast, this does not exist in distributed social networks which are formed by interlinked, distributed profiles[1]. Therefore, skill endorsements in profiles are also inherently distributed over the network. This requires to ensure discovery, comparability and description of skills across organization boundaries. Each skill endorsement needs to be described properly to facilitate comprehension and avoid misunderstandings. To allow adding further skill information, both skill set and skill descriptions need to be accessible, extensible and linkable. So, persons can easily refer to descriptions of the skill endorsements they made. When Alice claims she has a skill, this endorsement must be associated to the person making the claim i.e., to her. This is necessary because information a person produces, belongs to her. To persist skill endorsements associated to the person stating them, they need to be stored and connected with the person's identity. This enables persons and machines to detect skill endorsements, provided that relevant data is accessible in an easy-to-process manner. To achieve this objective, we need to deliver 1) an extensible description for each skill, 2) a way to attach all kinds of skill endorsements to persons, 3) a place to store each person's skill endorsements, and 4) a procedure to find experts in distributed social networks based on skills.

Objective 2: To increase credibility of skill self-claims. Skill self-claims are the most basic form of skill endorsements. Persons can use them to declare

[1] Distributed profiles are documents, which are accessible from different URLs and hosted on different servers, referencing each other. They describe persons.

that they have a certain skill set. So, Alice can claim she has a specific skill like C# programming. To increase the credibility of self-claimed skills, other persons should be enabled to affirm them. Skill affirmations allow persons to testify that someone they know has a specific skill. This affirmation may be based on past collaboration where certain skills were involved and demonstrated. For example, Bob can affirm Alice's C# skill because he worked with her on a project requiring this particular skill. In endorsing someone's skill, the endorser uses his own reputation to give more weight to the endorsee's claimed skill. This contributes to increasing the credibility of claimed skills. Achieving this objective requires delivering a procedure that allows persons to affirm self-claimed skills.

Objective 3: To allow assigning skills the endorsee did not consider. In many respects skill assignments are similar to skill affirmations. A skill assignment suggests a person, like Alice, to claim a skill she has not considered so far. As an example, Bob knows Alice very well. So, he might assign Alice the HTML skill she did not think of. While skill affirmations rely on prior skill self-claims, skill assignments do not. For achieving this objective, we need to deliver a procedure that allows persons to assign skills that have not been self-claimed beforehand.

Objective 4: To prevent unwanted skill endorsements. Centralized social networks can easily incorporate the concept of skill affirmations and skill assignments. They form a single point of truth. The skill endorsements are part of the database of the networking platform. Unless integrity of the data stock has been violated, it is impossible for Mallory to claim negative skills upon Alice. That is, the endorsee needs to self-claim skills beforehand or confirm an assignment.

Adopting this policy to distributed social networks without a central data base is more complicated. First, we must avoid maliciously negative affirmations and assignments. Otherwise, Mallory could affirm negative skills to damage the Alice's reputation by publicly claiming Alice has an "incompetence" skill. Second, persons might be found by expert finding systems due to outdated affirmations of skills they deliberately removed from their profiles. For example, an engineer who has been working for arms industry but now decided against this branch removes corresponding skills from his profile. Distributed expert finding should not consider outdated skill endorsements. Therefore, we need to strive for an agreement between the endorser and the endorsee. As a side effect, this would also contribute to increasing the credibility of skill claims.

Objective 5: To express skill endorsements in distributed user profiles. When Alice claims she has a certain skill set, this information must be recognizable by all authorized members of the distributed social network. The same holds true, when Bob endorses a skill of Alice or when Charlie makes a skill assignment. All three kinds of skill endorsements differ in who is claiming which skill for whom. Thus, each skill endorsement consists of three basic elements: endorser, skill and endorsee. So, a vocabulary able to express such triples in a unified and linkable manner would allow covering all kinds of skill endorsements. Associating and storing skill endorsements with the person claiming them, as suggested in Objective 1, requires delivering a vocabulary for specifying skill endorsements.

Objective 6: To incorporate skill endorsements in distributed expert finding. Achieving Objective 1 fulfills the basic requirements to incorporate skill endorsements in distributed expert finding. The expert finder needs to compare all skill endorsements associated to a candidate with the skills required for a task. For determining a person's suitability for a case, distributed expert finding must consider all kinds of skill endorsements. This assists Casey in deciding about assigning a task to Alice, Bob etc.. To achieve this objective, we need to deliver 1) a method to compare skill endorsements with case requirements and 2) a ranked list of experts fitting to the case requirements.

Objective 7: To facilitate a differentiated consideration of skill endorsements. Case owners benefit from an extensive knowledge about a candidate's suitability for a case. Taking all kinds of skill endorsements into account would enable Casey to gain a rich picture of each candidate's capabilities. Depending on the quantity and quality of a personal social network, the number of skill endorsements differs from person to person. For example, Alice's many social connections also entail many skill affirmations and assignments. The number of skill endorsements could be one criterion for Casey. She knows, however, that this would discriminate persons who have fewer or less diligent social connections.

Distributed expert finding could address such issues by statically weighting each kind of skill endorsement differently. This would, however, reduce adaptability of expert finding and favor persons who share similar characteristics. To preserve customizability, distributed expert finding has to enable adaptably factoring in all kinds of skill endorsements. So, Casey could weight skill self-claims more than skill affirmations or assignments. This is in line with Objectives 2 to 4.

3 Expert Finding with CRAWL

In this section we describe how CRAWL [15] assists expert finding in distributed social networks. The scenario shown in Figure 3 demonstrates our approach. Casey works as a second-level-support worker for a software development company. A key customer reports a bug in a software product developed by the company. Casey is responsible for the handling of this support case. She uses an ACM system to assist her work. As she investigates the problem, she defines several goals and asks experts from the third-level-support department to contribute. At some point during the analysis of the bug, a detailed profiling is required to rule out concurrency issues. However, there is no expert on this topic available. To assist Casey in finding a person with the required expertise, CRAWL facilitates the following workflow (cf. numbers in Figure 3):

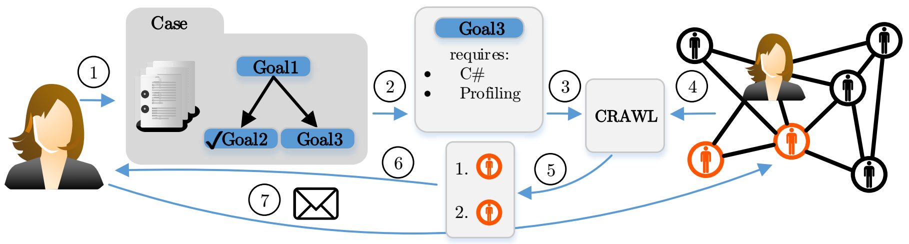


Fig. 3. CRAWL Overview

1. Casey adds a corresponding goal to the case.
2. Casey defines requirements (e.g., C# and Profiling).
3. Casey starts CRAWL.
4. CRAWL traverses Casey's social graph.
5. CRAWL generates a list of eligible workers.
6. Casey selects the most suitable candidates.
7. Casey asks them for contribution to the goal.

Finding suitable workers requires a traversal of the requestor's social graph. This graph is established by `foaf:knows` connections in WebID profiles. WebID profiles are essential artifacts of the WebID identification approach. They contain an identity owner's personal data described in a machine-readable way using Linked Data. For this, WebID relies on several RDF-vocabularies such as FOAF. With WebID, users are enabled to globally authenticate themselves, connect to each other, manage their profile data at a self-defined place and specify customized views [23]. Users can rely on WebID identity providers for creating new WebID identities and managing their WebID profile data [24].

The traversal algorithm is implemented as a depth-limited breadth-first search. It dequeues a WebID URI identifying a person, retrieves the corresponding WebID profile, calculates the rating R, marks the WebID URI as visited and adds all unvisited WebID URIs referenced via `foaf:knows` and their depth value to the queue. The initial queue consists of the WebID URIs of the persons already involved in the case. A maximum depth is used due to the exponentially rising number of nodes in a social graph with increasing depth [13]. CRAWL allows for additional limits like the number of suitable candidates rated above a certain threshold. Following the rating, the WebID profile graph of the candidates is added to a triplestore. A statement containing the calculated rating is asserted into the graph. The final ordered list of rated candidates results from executing the SPARQL query shown in Listing 1.1 on the triplestore.

Sequential traversal of WebID profiles and rating calculation have a huge impact on performance due to the distributed nature of profiles. We addressed this issue by concurrency and caching of user profiles and skill descriptions [15].

```
SELECT ?candidate ?rating
WHERE { ?candidate a foaf:Person .
  ?candidate vsrcm:rating ?rating.
  FILTER( ?rating > ?minRating )}
ORDER BY DESC( ?rating )
```

Listing 1.1. SPARQL query for candidates

Having retrieved and rated a subset of the social graph, CRAWL presents a list of recommended candidates and contact information to the person initiating the search. This step allows for later extension to enable applying constraint criteria, e.g., filter candidates from a specific company or within the same country. CRAWL demonstrates the basic concept of expert finding in distributed social networks leveraging knowledge from profiles, case descriptions and Linked Open Data (LOD) [15]. Therefore it addresses Objective 1. Yet, it does not consider knowledge of people about people such as skill assignments and skill affirmations.

4 CRAWL·E: Extending CRAWL with Endorsements

In order to addresses all objectives from Section 1, we propose CRAWL·E which extends CRAWL with endorsements. The first part of this section introduces a vocabulary to express skill endorsements. Part two explains the expert finding algorithm and the integration of endorsements in the candidate rating.

4.1 Integrating Skill Endorsements in Distributed Profiles

In [15] we introduced a vocabulary to add skill self-claims to WebID profiles. Linked Data provides CRAWL with a large knowledge base for concepts describing skills. CRAWL references this data to describe existing experience for persons and experience required to achieve a case goal or contribute to it. In a WebID profile, the RDF property `vsrcm:experiencedIn` connects a `foaf:Person` with a URI which represents this person's experience in *something.* For referring to the actual skills URIs are used to reference concepts which are available as dbpedia[2] resources. With dbpedia being a central element of the linked open data cloud, this intends to increase the degree of reusability and extensibility of skill data.

To express endorsements, we reuse this vocabulary as seen in Listing 1.2. The important aspect to note is the distributed nature of profiles. An endorser has no write access to foreign endorsees' profiles.

As there is no specific platform or protocol defined for adding statements to WebID profiles, skill assignments have to be expressed in the endorser's own profile. Leveraging the RDF data model and FOAF vocabulary, CRAWL·E enables persons to add skill assignments to their WebID profiles. These skill assignments

[2] `http://dbpedia.org/`

```
@prefix endorser: <http://company.org/>.
@prefix endorsee: <http://minisoft.ru/>.

<endorser:bob> a foaf:PersonalProfileDocument;
    foaf:primaryTopic <endorser:bob#me>;
    foaf:title "Bob Endorser's WebID profile".

<endorser:bob#me> a foaf:Person;
    foaf:name "Bob Endorser";
    foaf:knows <endorsee:alice#me>;
    cert:key [a cert:RSAPublicKey ;
             cert:exponent 65537  ;
             cert:modulus "1234..."^^xsd:hexBinary].

<endorsee:alice#me> vsrcm:experiencedIn <dbp:Linux>,
                                        <dbp:Mysql>.
```

Listing 1.2. Skill assignment in endorser's WebID profile

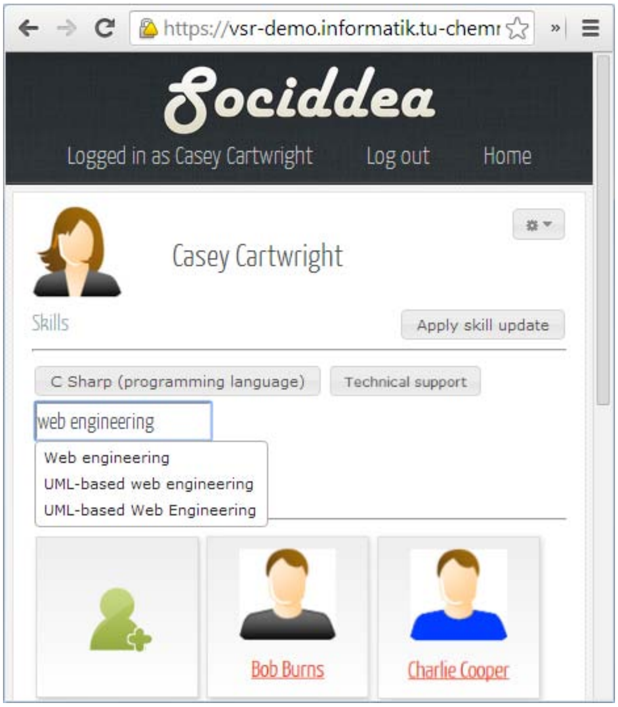

https://vsr-demo.informatik.tu-chemn
Sociddea
Logged in as Casey Cartwright
Log out
Home
Casey Cartwright
Skills
Apply skill update
C Sharp (programming language)
Technical support
web engineering
Web engineering
UML-based web engineering
UML-based Web Engineering
Bob Burns
Charlie Cooper

Fig. 4. Skill definition in Sociddea

reference the WebID URI of the endorsee who is connected to the endorser via `foaf:knows`.

Supporting users in specifying their expertise and case requirements, we exemplarily extended the user interfaces of Sociddea and VSRCM [15] to allow specifying skills using regular English words. We use prefix search of dbpedia lookup service to match user input against dbpedia resources. A list of skills is updated live as the user is typing. This is illustrated in Figure 4.

4.2 Extending Distributed Expert Finding to Leverage Skill Endorsements

This section describes how CRAWL·E incorporates endorsements in candidate rating. Figure 5 shows the traversal, rating and candidate recommendation which form steps 4 and 5 in Figure 3. The required skills s_{r0}, s_{r1}, s_{r2} of Goal3 and Casey's social graph are the input. In this example, Casey knows B and C. B and C know D, C knows E. CRAWL·E has already rated B with $R(B) = 15$, C with $R(C) = 0$ and D with $R(D) = 10$. To get the rating of E, the similarities between required skills and existing skills are calculated using linked open data. For a proof-of-concept, we use a prototypical rating function adapted from [16].

According to Objective 5, a skill endorsement is a triple (p_1, p_2, s) of endorser p_1, endorsee p_2 and skill s. A self-claimed skill can be represented as (p, p, s): by an endorsement with identical endorser and endorsee. A candidate c is described by E, the set of all endorsements regarding c in c's social graph as in Equation (1).

$$E = \{(p_1, p_2, s) | p_2 = c\} \tag{1}$$

S_S is the set of self-claimed skills by the candidate, S_O is the set of skills endorsed (assigned) by others and S_B is the set of skills claimed by the candidate and affirmed by others (both) (also cf. to Figure 1).

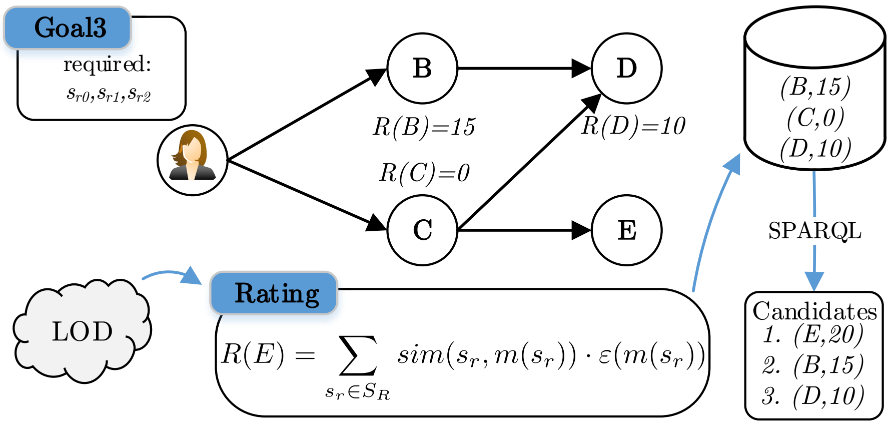


Fig. 5. Traversal, rating and recommendation

$$S_S = \{s | \exists (c, c, s) \in E\} \tag{2a}$$
$$S_O = \{s | \exists (p_1, c, s) \in E, p_1 \neq c\} \tag{2b}$$
$$S_B = S_S \cap S_O \tag{2c}$$
$$S = \{s | \exists (p_1, c, s) \in E\} = S_S \cup S_O \tag{2d}$$

CRAWL·E compares the set of required skills S_R to the set of skills S of each candidate as defined in Equation (2d). Both skill sets are represented by sets of dbpedia URIs. The similarity $sim(s_1, s_2)$ between two skills distinguishes different concept matches:

1. Exact Concept Match - URIs are identical ($s_1 = s_2$)
2. Same Concept As Match - URIs connected via `owl:sameAs` (s_1 `owl:sameAs` s_2)
3. Related Concept Match - URIs connected via `dbprop: paradigm`, `dcterms:subject`, `skos:narrower` etc.

This forms the similarity function in Equation (3).

$$sim(s_1, s_2) = \begin{cases} \sigma_1 & \text{if Exact Concept Match} \\ \sigma_2 & \text{if Same Concept As Match} \\ \sigma_3 & \text{if Related Concept Match} \\ 0 & \text{else} \end{cases} \tag{3}$$

These concept match types can easily be extended to facilitate an adapted rating. The basic idea is that each type yields a different similarity rating. For the moment, we use these values: $\sigma_1 = 10$ for 1), $\sigma_2 = 9$ for 2) and $\sigma_3 = 5$ for 3).

Per candidate, for each combination of a required skill $s_r \in S_R$ and a candidate skill $s_c \in S$, the similarity $sim(s_r, s_c)$ is computed according to Equation (3). As seen in Equation (5), to calculate the candidate rating R in CRAWL, only the skill with maximum similarity per required skill (from function $m : S_R \to S$ in eq. (4)) is considered. [15]

$$m(s_r) = s_c \Leftrightarrow sim(s_r, s_c) = \max_{s \in S} sim(s_r, s) \tag{4}$$

$$R_{\texttt{CRAWL}}(c) = \sum_{s_r \in S_R} sim(s_r, m(s_r)) \tag{5}$$

In CRAWL·E, an affirmed skill is given higher influence compared to a self-claimed skill. To accomodate this influence, we introduce the endorsement factor ε as defined in Equation (7). Our updated CRAWL·E rating function is shown in Equation (6).

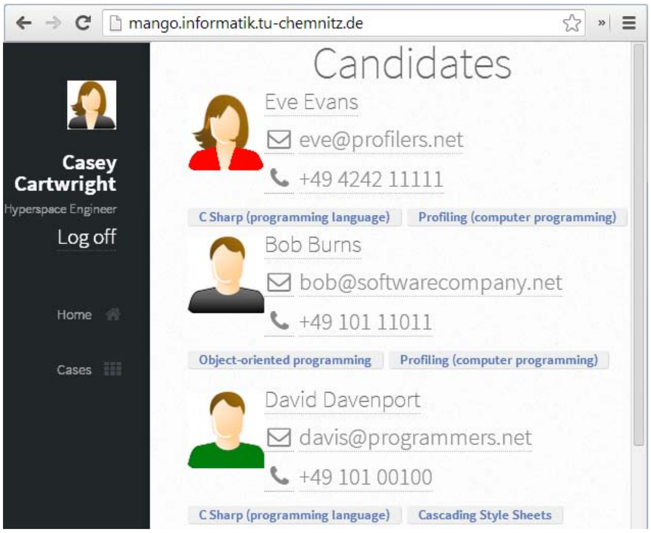
$$R_{\texttt{CRAWL·E}}(c) = \sum_{s_r \in S_R} sim(s_r, m(s_r)) \cdot \varepsilon(m(s_r)) \tag{6}$$

Let c be a candidate with the set of endorsements E. The set of all endorsements of the candidate skill s_c is defined by $E_{s_c} = \{(p_1, p_2, s) \in E | s = s_c\} \subseteq E$. With this, we define the endorsement factor using the skill sets defined in 2

$$\varepsilon(s_c) = \begin{cases} 1 & \text{if } s_c \in S_S \setminus S_B \\ \alpha\sqrt{|E_{s_c}|} & \text{if } s_c \in S_B \\ \beta\sqrt{|E_{s_c}|} & \text{if } s_c \in S_0 \setminus S_B \end{cases} \tag{7}$$

This factor distinguishes between the three types of skills: Self-claimed-only skills - from $S_S \setminus S_B$, skills that have been claimed by the candidate and endorsed by others - from S_B - and skills that have only been endorsed by others but are not stated in the candidate's profile - from $S_O \setminus S_B$. It yields 1 for a candidate skill without endorsements, i.e., no additional influence is given to self-acclaimed skills. Parameters α and β allow for adaption, currently we use $\alpha = 1.5$ and $\beta = 2$. To ignore unilateral skill assignments one can set $\beta \overset{!}{=} 0$.

The endorsement factor ε increases with the number of endorsements. However, the higher the number of endorsements, the slower the factor increases. This is to avoid overrating candidates with very high endorsement counts. Other function types such as a mirrored $1/x$ function are possible, too. We decided in favor of the square root function type, because it does not converge against a limit as there is no theoretical foundation to reason the limit. When the rating is finished, recommended candidates can be listed as in Figure 6.



Candidates

Eve Evans
eve@profilers.net
+49 4242 11111
C Sharp (programming language) Profiling (computer programming)

Bob Burns
bob@softwarecompany.net
+49 101 11011
Object-oriented programming Profiling (computer programming)

David Davenport
davis@programmers.net
+49 101 00100
C Sharp (programming language) Cascading Style Sheets

Fig. 6. Candidate recommendation in VSRCM

5 Evaluation

This section discusses the evaluation of our approach. We claim that CRAWL·E achieved the overall objective stated in Section 1 by considering all three kinds of skill endorsements for expert finding in distributed social networks. To prove this claim, we first outline the evaluation setup and then discuss our findings.

5.1 Evaluation Setup

To evaluate the extent to which CRAWL·E's objectives have been achieved, we chose the objective-based evaluation method. Event though a field experiment or a case study would allow for a profound review also, CRAWL·E's results highly depend on the underlying distributed social network. There are various characteristics to be considered, including total and average amount of social connections and of each kind of skill endorsements, richness of user profile data, and level of networking. While a prototypical implementation of CRAWL·E is publicly available, the adoption[3] by users has not yet reached a certain level. Such adoption is, however, required for conducting a field experiment and gaining both extensive and reliable evaluation results. Yet, we are convinced that enabling knowledge workers to find the right collaborators impacts the performance of future enterprises [5]. Thus, the value add by skill endorsements will soon trigger a demand to incorporate them into distributed worker profiles and expert finding.

An objective-based study is the most prevalent approach in program evaluation and is applicable to be performed internally by program developers [20]. As Stufflebeam describes, the "objectives-based approach is especially applicable in assessing tightly focused projects that have clear, supportable objectives" and that "such studies can be strengthened by judging project objectives against the intended beneficiaries' assessed needs, searching for side effects, and studying the process as well as the outcomes". For devising CRAWL we already defined objectives in Section 2. They are well-suited to be reused as evaluation criteria in this objective-based study. We incorporate the information collected during development and application of CRAWL and CRAWL·E, cf. Heil et al. in [15], to determine how well each operational objective was achieved. After outlining the evaluation setup, we discuss the findings for each objective in the following.

5.2 Discussion of Findings

To enable expert finding in distributed social networks, we follow the idea of finding suitable experts to invite. We think that searching for experts by utilizing personal social graphs is more beneficial compared to the open call approach discussed in Crowdsourcing research. Case owners intending to delegate tasks know their social network. So, they know whether a candidate fits the task description. CRAWL·E allows describing skills associated persons and requirements associated to cases. Rather than building our own database to manage skills and skill descriptions as typical for centralized social networks, our approach relies on the collective knowledge of dbpedia. In CRAWL·E, linking to a dbpedia resource refers either to a skill or to a requirement. Unlike related work, we do not want to restrict the number of options for skills and requirements. The availability of a description for a referred concept is not just a requirement in our approach, but also a good practice in general. dbpedia is a central part of the Linked

[3] With regard to the quantity of skill self-claims, skill affirmations and skill assignments per user and in total, and the use of the skill endorsement vocabulary in general.

Open Data cloud. So, resources are machine-readable through RDF and highly connected to each other. This allows for classification and association. The set of resources as well as each resource as such is extensible and maintained by a large community. When attaching skills to persons, we benefit from skills that are both referenced and identified by URIs. Similar to a skill URI pointing to a skill description, we use a WebID URI to refer to a person. Contrary to creating separate resources for storing each person's skill endorsements, we embedded them in machine-readable WebID profiles. As all skill endorsements a person, like Bob, issues belong to him, they also remain in his profile. With all concepts described using RDF, discovery and query become possible through interlinkage with URIs and retrieval using SPARQL. Thus, we delivered all four results necessary for achieving this objective.

To increase credibility of skill self-claims, we introduced the concept of skill affirmations. With a skill self-claim, a person describes that he possesses this skill. The credibility of skills self-claimed by a person, however, depends on the level of trust in this person. A skill self-claim which has been endorsed by someone else gains credibility depending on the trust in the endorsing person. In CRAWL·E, we assume that the more persons have endorsed a person for a particular skill, the more likely is this person to *really* possess the skill. Thus, the higher endorsed a certain skill self-claim is, the more influence it has on the candidate ranking.

To allow assigning skills the endorsee did not consider, our approach enables endorsers to claim skills a person has, without the person having to self-claim those skills beforehand. That is, it is not required for an endorsee to state such skills in their own WebID profiles. This is useful for instance to provide a more complete skill profile that includes information which the described person did not think of. Skill assignments available in distributed social networks can be exploited for various purposes including requests for adding assigned skills to own user profiles. While the increased expressiveness by skill assignments allowed achieving this objective, it comes at the cost of loosing control about what is being endorsed. We addressed this issue as explained in the following paragraph.

To prevent unwanted skill endorsements, we imposed the requirement that endorser and endorsee are bilaterally connected. This indicates that both persons *deliberately* know each other and, hence, accept each other's opinion. As an example, Charlie assigned a certain skill, like COBOL programming, to Alice some time ago. She followed Charlie's suggestion and claims this skill herself. So, Charlie's assignment became an affirmation. Alice was also endorsed by other persons for this skill, i.e., Bob or even Mallory. Today, she is not that interested in this topic anymore. Therefore, Alice does not want to be found for this skill. She removes her self-claimed COBOL skill. Thus, all affirmations become assignments. By excluding Mallory from her social network, all his affirmations and assignments are going to be ignored in CRAWL·E. That is, affirming or assigning a skill necessitates that both the endorsee and the endorser know each other. In addition to this approach, we enable avoiding malicious and outdated skill endorsements by

1) simply removing corresponding skill self-claims the endorsee made in his profile and 2) appropriate consideration within distributed expert finding.

To express skill endorsements in distributed user profiles, we had to find an alternative to what is known from centralized social networks. There, Bob can endorse Alice for a skill she did not think of and the platform triggers a notification to Alice. The endorsed skill will not be added to her profile unless she approves of it. This is not possible with distributed social networks. Alice' and Bob's WebID profiles are documents typically hosted on different servers, and identified and retrievable by URIs. There is no platform to trigger a notification to Alice let alone to allow Bob writing to her profile. Objectives 2 and 3 are inherently adverse to Objective 4 because there is no standard means of asking approval. We therefore delivered an RDF vocabulary to specify skill endorsements using only one RDF triple per skill definition, no matter if it is a skill self-claim, affirmation or assignment. Associating endorser, endorsee and endorsed skill, and storing the triple within the endorser's WebID profile allows for expressing all three kinds of skill endorsements. Due to RDF's flexible and extensible nature, skill endorsements can be attached to the endorser (for self-claims) or to one of his social connections expressed via `foaf:knows` (for affirmations and assignments).

To incorporate skill endorsements in distributed expert finding, CRAWL·E queries personal social networks for skill information, retrieves and processes the skill endorsements, and compares them with the case requirements before showing them a ranked list of suited candidates. Our approach hereby involves how we have addressed Objective 5 for query and retrieval, and Objectives 2 to 4 for comparing and ranking. As manual assessment by crawling personal social networks is time-consuming, our approach assists case owners in their recruitment tasks, but also leaves the final assessment decision to them. While CRAWL only considers self-claimed skills for the candidate recommendation, CRAWL·E also takes skill affirmations and assignments into account. However, all skill endorsements made by persons not knowing each other are ignored in this process. For comparing case requirements with all three kinds of skill endorsements, our approach computes the similarity between both concepts. Although CRAWL·E differentiates exact from similar or related concepts via different weights, finding precise and profound weights requires further empirical evaluation.

To facilitate a differentiated consideration of skill endorsements, we developed CRAWL·E in a parametrized way which allows to choose whether to include foreign endorsed skills or to operate on confirmationary endorsements only. By employing our approach, case owners can adjust the influence of unilaterally endorsed statements, i.e., which value a statement is given that potentially many others claim upon a person but which this person does not claim himself. To reduce the effect of unusually many skill endorsements per person through a large amount of diligent social connections, we introduced a function for only partially factoring in large accumulations of skill affirmations and assignments. Considering Equation (7), CRAWL·E facilitates fine-tuning and even excluding the impact skill affirmations and skill assignments have on distributed expert finding. Users can align the rating with their individual needs and preferences. Finding a more exact function type and evidenced-based default values for the parameters requires a larger empirical evaluation to be conduct in future work.

6 Related Work

Expert Finding has long been a research interest. For example, Becerra-Fernandez provides an overview on Web-based expert finding approaches in [3]. Like [2], many of them are based on information retrieval techniques. To achieve expert finding, they analyze document topics and connect them to the document authors. Perugini et al. surveys expert finding systems with a focus on social context including communication and blogs [19]. Particularly for the research domain, there are several approaches, e.g., by Xu et al. [25] or by Uddin et al. [22].

The approach described by Xu et al. in [25] is similar to CRAWL·E in that it unites social graphs with skill relationship semantics. To achieve this, network analysis on interlinked concept (expertise) and research (social) layer is employed. While this approach also considers hierarchical and correlation relationships in the expertise layer, tacit knowledge is used. CRAWL·E uses explicit knowledge from profiles and Linked Open Data, whereas [25, 2] extract information from unstructured text sources, [25] supported by WordNet. This works well in a specific domain like research, because characteristics of the domain can be exploited. For instance, citations and co-authorship can be analysed from publications [25].

By contrast, CRAWL·E is a generic approach not limited to a specific domain. None of the above approaches works in distributed social networks, nor do they explicitly consider endorsements.

In [6], Bozzon et al. present an expert finding method based on user's activities in centralized social networks like Facebook, Twitter etc. It analyzes social resources directly related (e.g. tweets, likes) or indirectly related (e.g. posts of liked pages) to persons. This approach employs text analysis: entity recognition for skills is performed on the resources, they are identified with Wikipedia URIs. CRAWL·E by contrast targets distributed social networks, uses explicit expertise information, supports skill endorsements and leverages linked data.

In spite of their benefits, skill endorsements have not yet gained much attention in research. Platforms like LinkedIn[4] and ResearchGate[5] have successfully included them. Within the first six month, more than 2 billion endorsements were created on LinkedIn allowing for interesting analysis [14]. Donston-Miller states in [10] that endorsements provide a streamlined version of a resume and can reduce the risk of hiring new personnel. Also quality aspects should be considered in addition to mere quantity (endorsement count) measures. Doyle also mentions in [11] the problem of unwanted endorsements and argues that getting the "right" endorsements is important. While Berk suspects that LinkedIn is using endorsement data in its secret search algorithm [4], there is not much public

[4] `http://www.linkedin.com/`
[5] `http://www.researchgate.net/`

information available about skill endorsements in expert finding. Even if current platforms are internally implementing this, the major difference to CRAWL·E is the central nature of these social networks.

Pérez-Rosés et al. presents in [18] an approach which combines social graphs with skill endorsements. It uses an undirected social graph and a directed endorsement graph per skill. Skill relationships like correlation or implied skills are considered. The PageRank algorithm is applied to a deduced graph. Its deduction matrix is similar to the similarity matrix in CRAWL·E. However, the definition of the deduction matrix is an open problem whereas we get the values of the matrix leveraging Linked Open Data. Friendship-like bilateral relationships between social network members are assumed, while the `foaf:knows` semantics employed in CRAWL·E allows for unilateral relationships. Unlike CRAWL·E, this work focuses on social graph analysis, lacks a complete expert finding workflow, and does not support distributed social networks.

Our approach is an application of the *social routing principle* [12] to the ACM domain. Unlike task delegation through an open call known from Crowdsourcing research [7], we follow the idea of inviting suitable experts to contribute to a case by utilizing social graphs. The *conceptual routing table* described by Dustdar and Gaedke is formed by `foaf:knows` statements and contact info in WebID profiles.

7 Conclusions and Future Work

In this work, we presented the CRAWL·E approach leveraging distributed endorsements for finding eligible workers to contribute to ACM cases. It comprises a vocabulary for skill endorsements in WebID profiles, a method for traversing distributed social networks based on foaf:knows relationships and an adaptable rating function for WebID profiles. We demonstrated CRAWL by implementation based on the WebID identity provider and management platform Sociddea and the case management system VSRCM.

Our future research interest will be to consider not only endorsement quantity, but also quality. If a renowned expert endorses someone else for his very own field, his endorsement should be given more weight compared to endorsements of less renowned persons. This needs considering the endorsements of the endorsers in addition to the endorsements of the candidate to rate. Empirical data and machine learning can be used to provide adapted parameters. Providing Distributed Expert Finding as a Service is desirable to enable easy integration in other systems. For this, an endpoint structure and protocol must be defined.

References

[1] Appelquist, D., et al.: A Standards-based, Open and Privacy-aware Social Web: W3C Incubator Group Report. Tech. rep., W3C (2010)
[2] Balog, K., Azzopardi, L., de Rijke, M.: Formal models for expert finding in enterprise corpora. In: Proceedings of the 29th Annual International ACM SIGIR Conference, pp. 43–50. ACM, New York (2006)

[3] Becerra-Fernandez, I.: Searching for experts on the Web: A review of contemporary expertise locator systems. ACM TOIT 6(4), 333–355 (2006)
[4] Berk, R.A.: Linkedin Triology: Part 3. Top 20 Sources for Connections and How to Add Recommendations. The Journal of Faculty Development 28(2), 1–13 (2014)
[5] Bider, I., Johannesson, P., Perjons, E.: Do workflow-based systems satisfy the demands of the agile enterprise of the future? In: La Rosa, M., Soffer, P. (eds.) BPM Workshops 2012. LNBIP, vol. 132, pp. 59–64. Springer, Heidelberg (2013)
[6] Bozzon, A.,et al.: Choosing the Right Crowd: Expert Finding in Social Networks Categories and Subject Descriptors. In: Proceedings of the 16th International Conference on Extending Database Technology, New York, NY, USA, pp. 637–348 (2013)
[7] Brabham, D.C.: Crowdsourcing as a Model for Problem Solving: An Introduction and Cases. Convergence: The International Journal of Research into New Media Technologies 14(1), 75–90 (2008)
[8] Clair, C.L., Miers, D.: The Forrester Wave^{TM}: Dynamic Case Management, Q1 2011. Tech. rep., Forrester Research (2011)
[9] Davenport, T.H.: Rethinking knowledge work: A strategic approach. McKinsey Quarterly (2011)
[10] Donston-Miller, D.: What LinkedIn Endorsements Mean To You (2012), `http://www.informationweek.com/infrastructure/networking/what-linkedin-endorsements-mean-to-you/d/d-id/1106795`
[11] Doyle, A.: How To Use LinkedIn Endorsements (2012), `http://jobsearch.about.com/od/linkedin/qt/linkedin-endorsements.htm`
[12] Dustdar, S., Gaedke, M.: The social routing principle. IEEE Internet Computing 15(4), 80–83 (2011)
[13] Goel, S., Muhamad, R., Watts, D.: Social search in "small-world" experiments. In: Proceedings of the 18th International Conference on World Wide Web, WWW 2009, pp. 701–710. ACM, New York (2009)
[14] Gupta, S.: Geographic trends in skills using LinkedIn's Endorsement feature (2013), `http://engineering.linkedin.com/endorsements/geographic-trends-skills-using-linkedins-endorsement-feature`
[15] Heil, S., et al.: Collaborative Adaptive Case Management with Linked Data. To appear in WWW 2014 Companion: Proceedings of the 23rd International Conference on World Wide Web Companion, Seoul, Korea (2014)
[16] Lv, H., Zhu, B.: Skill ontology-based semantic model and its matching algorithm. In: CAIDCD 2006, pp. 1–4. IEEE (2006)
[17] Mundbrod, N., Kolb, J., Reichert, M.: Towards a system support of collaborative knowledge work. In: La Rosa, M., Soffer, P. (eds.) BPM Workshops 2012. LNBIP, vol. 132, pp. 31–42. Springer, Heidelberg (2013)
[18] Pérez-Rosés, H., Sebé, F., Ribó, J.M.: Endorsement Deduction and Ranking in Social Networks. In: 7th GraphMasters Workshop, Lleida, Spain (2013)
[19] Perugini, S., Goncalves, M.A., Fox, E.A.: A connection-centric survey of recommender systems research. Journal of Intelligent Information Systems 23(2), 107–143 (2004)
[20] Stufflebeam, D.: Evaluation Models. New Directions for Evaluation 2001(89), 7–98 (2001)
[21] Swenson, K.D.: Position: BPMN Is Incompatible with ACM. In: La Rosa, M., Soffer, P. (eds.) BPM Workshops 2012. LNBIP, vol. 132, pp. 55–58. Springer, Heidelberg (2013)
[22] Uddin, M.N., Duong, T.H., Oh, K.-j., Jo, G.-S.: An ontology based model for experts search and ranking. In: Nguyen, N.T., Kim, C.-G., Janiak, A. (eds.) ACIIDS 2011, Part II. LNCS, vol. 6592, pp. 150–160. Springer, Heidelberg (2011)

[23] Wild, S., Chudnovskyy, O., Heil, S., Gaedke, M.: Customized Views on Profiles in WebID-Based Distributed Social Networks. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 498–501. Springer, Heidelberg (2013)
[24] Wild, S., Chudnovskyy, O., Heil, S., Gaedke, M.: Protecting User Profile Data in WebID-Based Social Networks Through Fine-Grained Filtering. In: Sheng, Q.Z., Kjeldskov, J. (eds.) ICWE Workshops 2013. LNCS, vol. 8295, pp. 269–280. Springer, Heidelberg (2013)
[25] Xu, Y., et al.: Combining social network and semantic concept analysis for personalized academic researcher recommendation. Decision Support Systems 54(1), 564–573 (2012)

Cross Publishing 2.0: Letting Users Define Their Sharing Practices on Top of YQL

Jon Iturrioz, Iker Azpeitia, and Oscar Díaz

ONEKIN Group, University of the Basque Country, San Sebastián, Spain
`{jon.iturrioz,iker.azpeitia,oscar.diaz}@ehu.es`

Abstract. One of Web2.0 hallmarks is the empowerment of users in the transit from consumers to producers. So far, the focus has been on content: text, video or pictures on the Web has increasingly a layman's origin. This paper looks at another Web functionality, cross publishing, whereby items in one website might also impact on sister websites. The *Like* and *ShareThis* buttons are forerunners of this tendency whereby websites strive to influence and be influenced by the actions of their users in the websphere (e.g. clicking on *Like* in site A impacts a different site B, i.e. *Facebook).* This brings cross publishing into the users' hands but in a "canned" way, i.e. the 'what' (i.e. the resource) and the 'whom' (the addressee website) is set by the hosting website. However, this built-in focus does not preclude the need for a 'do-it-yourself' approach where users themselves are empowered to define their cross publishing strategies. The goal is to turn cross publishing into a crosscut, i.e. an ubiquitous, website-agnostic, do-it-yourself service. This vision is confronted with two main challenges: website application programming interface (API) heterogeneity and finding appropriate metaphors that shield users from the technical complexities while evoking familiar mental models. This work introduces *Trygger*, a plugin for *Firefox* that permits to define cross publishing rules on top of the *Yahoo Query Language* (*YQL*) console. We capitalize on *YQL* to hide API complexity, and envision cross publishing as triggers upon the *YQL*'s virtual database. Using *SQL*-like syntax, *Trygger* permits *YQL* users to specify custom cross publishing strategies.

Keywords: Data sharing, YQL, triggers, cross publishing, web service.

1 Introduction

People interaction on the Web has drastically evolved in the last few years. From business-to-consumer (B2C) interactions, the Web is now a major means for direct person-to-person interaction through social-media websites (hereafter referred to as "websites"). *Facebook*, *WordPress*, *Wikipedia* showcase this new crop of media where content is provided for and by end users. As the new content providers, end users are also amenable to define their own cross publishing strategies. Traditionally, cross publishing means allowing content items in one website to also appear in sister websites where the sharing strategies are set by the web masters. However, when the source of the content is the user (like in social-media websites), cross publishing should also be made 2.0, i.e. amenable to be defined by the end user. A first step is the *Like* and *ShareThis* buttons which let users *enact* cross publishing whereby the website hosting the button (e.g. pictures in *Flickr*) also impacts the content of a sister website (e.g. *Flickr* provides sharing for *Facebook*, *Twitter* and *Tumblr*). The enactment is up to the user. However, the 'what' (i.e. the resource) and 'to whom' (the addressee website) are still built-in.

Current mechanisms for cross publishing (e.g. the *Like* button) might fall short in some scenarios. First, minority websites will be rarely offered *"ShareThis"* links within main players' websites like *Twitter* or *Youtube*. Second, sharing might be conducted in a routinary way (e.g. every time I upload a presentation in *Slideshare*, communicate it to my followers in *Twitter*). Rather than forcing users to click the *Twitter* icon when in *Slideshare*, this situation can be automatized through a "sharing rule". Third, the rationales behind sharing are not explicit but kept in the user's mind. Sharing might be dictated by some resource characterization that could be captured through a "sharing rule" (e.g. create a task in my *ToDo* account when I tag a bookmark in *Delicious* with "toread").

In a previous approach [10], we provide the infrastructure for web masters to define cross publishing. However, as the previous scenarios highlight, it is difficult for a webmaster (e.g. the *Flickr*'s one) to foresee the different settings in which its resources will need to be shared and talked about (using tweets, posts, articles, etc.). This calls for cross publishing to become a crosscut website-agnostic end-user service. Web2.0 demonstrates that end users are ready [7] and willing to adapt their Web experience, if only the tools become available that make it sufficiently easy to do so (e.g. [6]). Cross publishing should not be an exception.

This work aims at ascertaining to which extend this vision is feasible both technically and humanly. From a technical perspective, we develop ***Trygger***, a *Firefox* plugin, that allows to define sharing rules as services over websites. The human factor is being considered by conceiving websites as database tables where sharing is realized as database-like triggers over these tables. By taping on a familiar representation (the relational model), our hope is to easy adoption. As an example, consider *Twitter* and *Facebook* as hosting a table of "tweets" and "wall posts", respectively. Users can be interested in publishing into their *Facebook* wall, tweets from their colleagues that contain the hashtag *#ICWE2014*. The *Trygger* expression will look something like: ***ON*** *INSERT a new tweet LIKE "%#ICWE2014%"* ***DO*** *INSERT a message (tweet permalink) INTO my wall.* This is the vision *Trygger* strives to accomplish. Compared with related work, the distinctive aspects of this approach include:

– Cross publishing is conceived as a crosscut service. We regard cross publishing as an idiosyncratic, traversal activity to be conducted in the cloud but managed locally. Users can define their own sharing rules, not being limited to those hardwired into the websites.

– Cross publishing is specified as triggers (Section 3 & 4). A familiar *SQL*-like syntax is offered from expressing sharing rules. We heavily rely on *Yahoo Query Language* (*YQL*) [13]. *YQL* provides a database vision of websites' APIs. *Trygger* extends this vision by providing a trigger-like mechanism for sharing over disparate websites.
– Cross publishing is implemented via a reactive architecture (Section 5). We provide a loosely coupled architecture for distributed trigger management. The implementation relies on existing standards and protocols, and it is entirely *HTTP*-based.

Trygger is fully realized as a *Firefox* plugin. This plugin, the working examples and a brief installation guide can be downloaded from *http://www.onekin.org /trygger/*

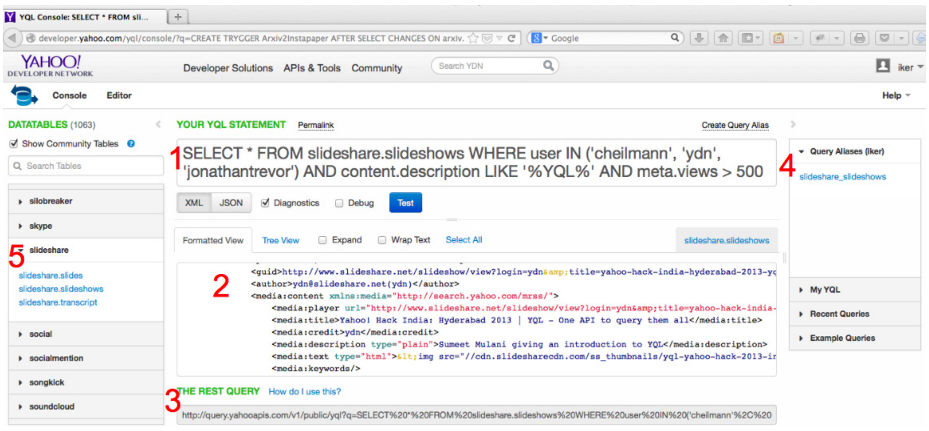
2 A Brief on YQL

A tenant of current Web development is the release of data through accessible APIs. The problem is that APIs are as heterogeneous as the applications they support. As a practitioner of the Programmable Web puts it "the main difficulty with APIs in general is that they require the developer to create and remember a number of cryptic URLs via which certain data can be retrieved from the API"[1]. *Yahoo Query Language* (*YQL*) tries to address this by converting API calls into SQL-like commands, which are somewhat more human readable. Figure 1 (1) illustrates *YQL*'s *SELECT* to access the *Slideshare* API but now conceived as the table *slideshare.slideshows*. To realize this metaphor, *YQL* offers both a service and a language.

***YQL* Language.** So far, *YQL* offers *SELECT, INSERT* and *DELETE* statements that, behind the curtains, invoke the corresponding API methods. *YQL* alleviates programmers from details concerning parameter passing, credential handling or implicit iterations (for joins). The mapping from the *YQL* syntax to API requests is achieved through the so-called **"Open Data Tables"** (ODT). ODTs hold all the intricacies of the underlying APIs. It is worth noticing that this sample query (Figure 1 (1)) actually conducts three API requests, one for each user ('cheilmann', 'ydn', 'jonathantrevor'). *YQL* conceals these interactions, enacts the three requests, combines the outputs, and filters out those results based on the query parameters. In this way, *YQL* off-loads processing that programmers would normally do on the client/server side to the *YQL* engine.

***YQL* Service**. This is realized through the *YQL* console[2]. The console permits to preview the query, and obtain the REST query counterpart. This query that can then be bookmarked or included in the programs. This interface contains the following main sections (see Figure 1): (1) the *YQL* statement section is

[1] `http://blog.programmableweb.com/2012/02/08/like-yql-try-it-with-the-programmableweb-api/`
[2] `http://developer.yahoo.com/yql/console/`



```
SELECT * FROM slideshare.slideshows WHERE user IN ('cheilmann', 'ydn',
'jonathantrevor') AND content.description LIKE '%YQL%' AND meta.views > 500
```

```
<guid>http://www.slideshare.net/slideshow/view?login=ydn&amp;title=yahoo-hack-india-hyderabad-2013-yq
<author>ydn@slideshare.net(ydn)</author>
<media:content xmlns:media="http://search.yahoo.com/mrss/">
    <media:player url="http://www.slideshare.net/slideshow/view?login=ydn&amp;title=yahoo-hack-india-
    <media:title>Yahoo! Hack India: Hyderabad 2013 | YQL - One API to query them all</media:title>
    <media:credit>ydn</media:credit>
    <media:description type="plain">Sumeet Mulani giving an introduction to YQL</media:description>
    <media:text type="html">&lt;img src="//cdn.slidesharecdn.com/ss_thumbnails/yql-yahoo-hack-2013-ir
    <media:keywords/>
```

```
http://query.yahooapis.com/v1/public/yql?q=SELECT%20*%20FROM%20slideshare.slideshows%20WHERE%20user%20IN%20('cheilmann'%2C%20
```

Fig. 1. *YQL* console

where *YQL* queries are edited, (2) the results section displays the output of the query, once the source Web service was enacted, (3) the REST query section provides the URL for *YQL* queries, (4) the queries section gives access to queries previously entered, and (5) the data tables section lists all the APIs that can be accessed using *YQL*.

The bottom line is that SQL-like syntax becomes the means to simplify Web development based on open APIs. Open APIs are the enablers of cross publishing. Therefore, and akin to *YQL*'s teachings, we aim at making cross publishing scripts look like the development of DB scripts. We build upon the similitude of SQL's triggers and cross publishing as for keeping in sync data repositories, let these be database tables or website silos, respectively. Details follow.

3 Cross Publishing Scripts: The Notion of *Trygger*

Supporting cross publishing as a service implies the monitoring of a website account so that actions on this account ripple to other accounts. This can be described as event-condition-action rules. Rules permit to enact small pieces of code ('DO') when some update operation is conducted ('ON') provided some condition is met ('WHEN'). An example follows:

ON a new tweet is uploaded in account ('Twitter', 'oscar')
WHEN new tweet.status LIKE "%ICWE2014%"
DO create a wall post in account ('Facebook', 'iker')

Akin to the *YQL* metaphor, we use *SQL*-like syntax for triggers to specify cross publishing rules (hereafter referred to as "tr**y**ggers"). The whole idea is to tap into the existing ODT repository. Indeed, most of the ODT tables used throughout the paper were provided by the *YQL* community. Even so, *YQL* permits ODTs external to the system to be operated upon through the *USE* clause. This clause just provides the URL that holds the ODT definition. From then on, no difference exists in handling ODTs kept outside *YQL* boundaries. Similarly, the *ENV* clause offers a way to access hidden data (see section 5). Figure 2 outlines the *trygger* syntax. Next paragraphs address each of the *trygger* clauses.

```
{ENV <access_key>;}*
{USE <url> AS <odtTable>;}*
CREATE TRYGGER <tryggerName>
AFTER SELECT CHANGES ON <odtTable>
WHEN <yqlFilter>
BEGIN
{ [XMLNAMESPACES (<url> AS <alias>)] <yqlStatement>  |  <jsCode> }
END
```

Fig. 2. *Trygger*'s syntax. *Tryggers* are based on *YQL*'s ODT tables.

The Event. In *SQL*, events are risen by insertions, deletions or updates upon database tables. Since *YQL*'s ODT handle both insertions and deletions, it could be possible to define *trygger* events upon *YQL* operations. Notice however, that *YQL* is just another client built on top of somewhere else's APIs. These APIs can be accessed by other clients, and hence, "the tables" can be changed by applications other than *YQL*. Monitoring only *YQL* operations would make these other changes go unnoticed. Therefore, the only way to be aware of insertions is by change monitoring. Hence, the *trygger* event is *"after_ select_ changes"* which happens when two consecutive pollings incrementally differ. No "statement" *tryggers* are supported. Only "for each row", i.e. each new tuple causes the enactment of the *trygger*. The event payload refers to this new tuple, kept in the system variable *NEW*.

The Condition. In *SQL*, conditions check the event payload, i.e. the new/old tuple being inserted/deleted. Likewise, *tryggers* can check conditions in the new "ODT tuples" being inserted. Each new tuple causes the enactment of the trigger only if the conditions are satisfied.

The Action. *SQL* actions stand for procedures that are stored in the database. Procedures can be atomic (i.e. a single statement) or full-fledged programs (described using e.g. PL/SQL or Java). Back to *Trygger*, actions can be either atomic or composed. Atomic actions comprise a single statement, specifically, any "insert/update/delete" statement defined upon an ODT table. Action parameters can refer to either constants or *NEW*. However, single-statement actions fall short in numerous scenarios, namely: when payload parameters need some processing before being consumed; when data kept in other ODT tables is to be retrieved; when more than one ODT table needs to be changed[3]. This leads to composed actions, i.e. programs. Akin to *YQL*, the programming language is JavaScript. *Trygger*'s JavaScript permits both *YQL* actions and access to the

[3] Action statements are enacted as independent entities. So far, *Trygger* does not support transaction-like mechanisms.

```
A)  CREATE TRYGGER Twitter2Facebook
    AFTER SELECT CHANGES ON twitter.search
    WHEN q in ("from:iturrioz", "from:oscaronekin") AND text LIKE "%#ICWE2014%"
    BEGIN
    INSERT INTO facebook.setStatus (uid, status, access_token)
    VALUES ("689274514", NEW.results.text.*, "254|2.AQY00514|FbS4U_w")
    END

B)  CREATE TRYGGER Arxiv2Instapaper
    AFTER SELECT CHANGES ON arxiv.search
    WHEN search_query = "ti:YQL" AND journal_ref LIKE "% SPRINGER %"
    BEGIN
    XMLNAMESPACES ("http://www.w3.org/2005/Atom" AS atom)
    INSERT INTO instapaper.unread (username, password, title, selection, url)
    VALUES ("oscaronekin@gmail.com","12pass34", NEW.atom:entry.atom:title.*,
                  NEW.atom:entry.atom:summary.*, NEW.atom:entry.atom:id.*);
    END
```

Fig. 3. A) *Twitter2Facebook trygger. NEW* refers to the ODT tuple being inserted since the last time the *twitter.search* table was pulled. B) *Arxiv2Instapaper trygger.*

system variable NEW. Next section illustrates this syntax throughout different examples[4].

4 Trygger at Work

Cross Publishing from *Twitter* to *Facebook*. *Facebook* launched *SelectiveTwitter*[5] whereby tweets ending in the hashtag *#fb* are directly propagated to the user's *Facebook* status. This application can be conceptually described as a cross publishing rule: "**on** introducing a tweet that contains *#fb*, **do** update my *Facebook* wall". *Facebook* developers were forced to provide a generic hashtag (i.e. *#fb*) to accommodate no matter the user. However, users can certainly be interested in monitoring *domain-specific* hashtags. For instance, when in a conference (identified through a hashtag, e.g. #ICWE2014), users might be interested in tracking tweets referring to *#ICWE2014* from the *Twitter* accounts of some attendees. Unlike the *#fb* case, this approach does not force the user's colleagues to introduce the *#fb* hashtag but a tag that might already exist. It is also domain-specific as for attaching distinct sharing behaviour to different hashtags. Figure 3-A provides a *trygger* that supports domain-specific hashtag tracking. We resort to two ODT tables, *"twitter.search"* and *"facebook.setStatus"* that support selections and insertions on *Twitter* and *Facebook*, respectively.

This rule is triggered if the delta of two consecutive selects on *twitter.search* is not empty. This ODT table includes two columns: *"q"* and *"text"* that keep the username of the *Twitter* account and the tweet message, respectively.

[4] Used ODT's can be consulted at `http://github.com/yql/yql-tables` or in the YQL console.

[5] `http://www.facebook.com/selectivetwitter`. To date, this enhancement obtains 3.6 out of 5 review summary based on 2,666 reviews.

The *trygger* conditions checks first whether *"q"* holds either *"from:iturrioz"* or *"from:oscaronekin"*, and second, if the *"text"* contains the string *"#ICWE2014"*. If met, the *trygger*'s action results in a new tuple being inserted into the *facebook.setStatus* table. The newly created tuple (i.e. *uid, status, access_ token*) is obtained from the *NEW* variable and the credentials for the *Facebook* account.

Cross Publishing from Arxiv to Instapaper. *Arxiv*[6] is an online archive for electronic preprints of scientific papers. *Instapaper*[7] is a neat tool to save web pages for later reading. In this example, the *trygger* will monitor new preprints in *Arxiv* published in any *"Springer"* journal that contains *"YQL"* in the title. On detecting such preprints, the *trygger* will create a new entry in the *Instapaper* account of *"oscaronekin"*. This example involves two ODT tables: *arxiv.search* and *instapaper.unread* (see Figure 3-B). The former includes two columns, i.e. *search_ query* and *journal_ ref* that hold the title and the journal of the manuscript respectively.

On adding a new preprint in *arxiv.search*, the *trygg*er checks whether the search expression is *"ti:YQL"* and the manuscript venue contains "*Springer*". The interesting part is the *trygger*'s action. The action constructs a tuple out of NEW. Since "NEW.atom" holds an XML document, its content can be obtained using E4X dot [8]. To avoid clumsy expressions, an *XMLNAMESPACES* declaration is introduced (so does *SQL*).

5 Trygger Architecture

YQL console is a tester of third parties' APIs. This implies to use credentials (passwords, API keys,...) on the statements. *Tryggers* are composed of *YQL* statements, hence the credentials would be exposed to the *Trygger* service. This could prevent users from creating *tryggers*. However, *Trygger* does not analyze or decompose the statements in a *trygger*, the event part and the action part of a *trygger* are used as black boxes. Even though, users are able to hide credentials using the storage capabilities offered by *YQL*[9]. *YQL* allows to set up key values for use within ODTs and to storage keys outside of the *YQL* statements including them on environment files. In these way, users are in control of their credentials and manage them directly on the *YQL* console (see Figure 4). As an example, let's analyze the action part of the *trygger* on Figure 3-A. The credential (i.e. *access_ token*) is explicitly shown (*254|2.AQY00514|FbS4U_ w*). The *SET* keyword on the *YQL* language binds a key with a value. So, the *SET access_ token = "254|2.AQY00514|FbS4U_ w" ON facebook.setStatus* instruction establish that the *254|2.AQY00514|FbS4U_ w* value is assigned to the *access_ token* field each

[6] `http://arxiv.org/`

[7] `http://www.instapaper.com/`

[8] ECMAScript for XML (E4X) is a programming language extension that adds native XML support to ECMAScript.
`http://en.wikipedia.org/wiki/ECMAScript_for_XML`

[9] See `http://developer.yahoo.com/yql/guide/yql-setting-key-values.html` for more information.

time the *facebook.setStatus* table is invoked. The *SET* instruction should be hidden in an environment file through the *YQL* console. The response is a public key (e.g *store://kgnRBearlKjAI4rBdRntdf*) to use the hidden data in the environment file. Therefore the action part on our working trygger is simplified as:

```
ENV 'store://kgnRBearlKjAI4rBdRntdf';
INSERT INTO facebook.setStatus (uid, status)
VALUES ("689274514", NEW.results.text.*);
```

This grounds *Trygger* to be supported using a client-side architecture, i.e. as a browser extension. Since *Trygger*'s first audience is the *YQL* community, the *Trygger* extension should be tightly integrated with the *YQL* console (not to be confused with the *YQL* system). That is, users should define *tryggers* using the same console they are accustomed to setting *YQL* requests: i.e. the *YQL* website. This implies the *Trygger* extension to be *locally* supported as an HTML wrapper upon the *YQL* website. To this end, we superimpose on the *YQL* console, GUI elements (e.g. buttons and panels) for *trygger* management. Interacting with these GUI elements will enact the services of server-side components that reside on the cloud, completely detached from the *YQL* system.

Database triggers are kept in the catalogue of the Database Management System (DBMS). *YQL* mimics the role of the DBMS as the container of ODT tables. This seems to suggest for *tryggers* to be held within the boundaries of *YQL*. However, some differences between *YQL* and DBMS advice a closer look. DBMSs are the only custodian of their data. Hence, DBMSs also act as event detectors since any table operations must go through the DBMS. By contrast, *YQL* is just another client built on top of somewhere else's data. This data can be updated by agents other than *YQL*. Therefore, the event detector can not stop at the ODT table but go down to the website. These observations unfold two divergent requirements. On one hand, we aim at a seamless integration with *YQL* so that users feel a unique experience. On the other hand, we require separate *trygger* management to detect events on monitored sites.

5.1 Deployment

The main components on the Trygger system complementing the *YQL* system are these (see Figure 4):

1. **The *Trygger* console.** It is a Web wrapper built on top of the *YQL* console. Implementation wise, this is achieved using *Greasemonkey*[10], a weaver that permits locally executed scripts to modify web pages on the fly. The wrapper provides local management for *tryggers*, i.e. creation, verification, deletion, enabling and disabling. Although the tryggers are send to the *Trygger* service a local copy is storage for rapid accessing and backing up.

[10] `http://www.greasespot.net/`

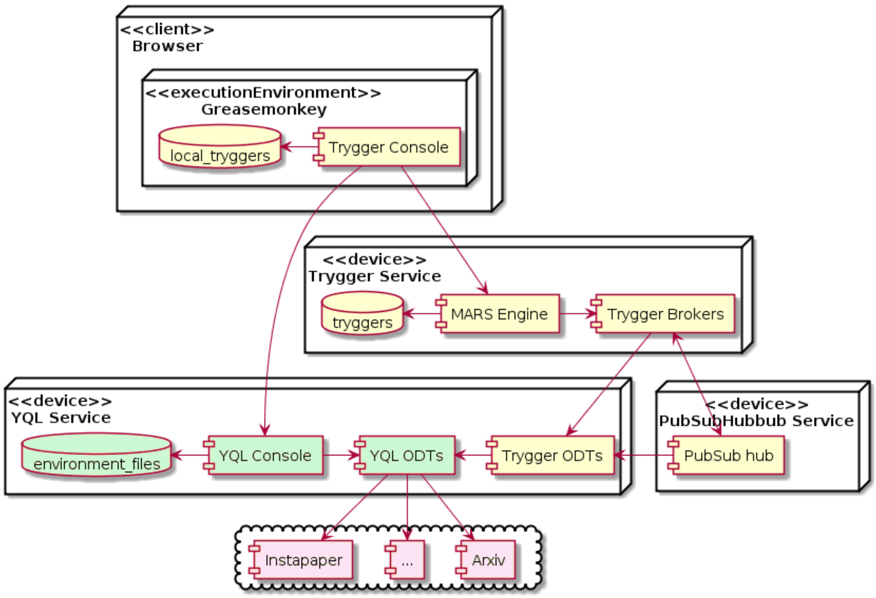


Fig. 4. Deployment diagram. In yellow color *Trygger* components, in green color *YQL* components and in pink color websites on the Internet

2. **The *Mars* Engine.** *Tryggers* are realized as Event Condition Action (ECA) rules. The *MARS* framework running into the *Trygger* service's machine is used as the rule engine [2]. Its functions include (see Figure 5): ECA rule registration (ECA engine), event occurrence signaling (Event Engine) and rule firing (i.e. rule's action enactment) (Action Engine). *MARS* provides the general framework for supporting ECA rules in a distributed environment. However, both events and actions are domain specific. Their realization and definition need to be externalized from *MARS* into so-called brokers.
3. **The *Trygger* Brokers**. The brokers are deployed in the *Trygger* service's machine. Two brokers are introduced (see Figure 5). The **Event Broker** monitors and generates events. So far, only one type of event is handled: changes in ODT table. Hence, the way to detect a change is by periodically polling the underlying website. For this purpose, we use the *PubSubHubbub* protocol [8]. On receiving the signal from the **PubSub hub**, the Event Broker generates an event occurrence for each NEW change. On event signaling, the *MARS* engine triggers the associated rules. Rule triggering is conducted through the **Action Broker**.
4. **The PubSub hub.** The *PubSubHubbub* service manages subscriptions to RSS data sources. The subscriber (i.e. the Event Broker) provides the publisher's feed URL (i.e. the *YQL* REST call to a ODT table) to be monitored as well as a callback URL to be called when the feed is updated. The hub

periodically polls the feed for new updates. On the feed being updated, the hub is noticed. This in turn propagates the feed to the subscriber callback endpoint: the Event Broker. There are some online *PubSubHubbub* services available on the Internet, for example, *Google PubSubHubbub Hub*[11] or *Superfeedr*[12]. However, we opt for deploy a dedicated service on the *Google App Engine* since the source code is freely downloadable[13].

5. **The *Trygger* ODTs**. They act as mediators from *MARS* to *YQL* (i.e. the **JSinterpreter ODT** shown in the Figure 5) and from *YQL* to PubSub hub (i.e. the **RSS-izator ODT** shown in the Figure 5)[14]. The former is a JavaScript snippet that knows how to process the code of the the rule's action part. The *Trygger* Action Broker requests this service providing as parameters the action code and the event occurrence. As for the RSS-izator, it resolves the format mismatch between the RSS required by the PubSub hub and the XML returned by *YQL*. The *trygger* ODTs reside into the *YQL*'s repository.

These components are already available in the cloud except the *Greasemonkey* wrapper. The wrapper needs to be locally installed as a *Firefox* plugin available at *http://www.onekin.org/trygger*. Next subsections outline how these components interact to support the main trygger life cycle stages: edition, verification, creation and enactment.

5.2 Interaction

The *trygger* definition and execution follows four steps:

***Trygger*Edition**. *Tryggers* are edited through the *YQL* console (see Figure 6). This console (1) detects the specification of a *trygger* in the *YQL* statement input box, and if this is the case, (2) provides a hyperlink for creating the *trygger* which is stored locally. So-stored *tryggers* can latter by managed through the new *TRYGGER ALIASES* tab (3). This tab displays a list of the locally kept *tryggers* together with icons for deletion, enabling/disabling and reading the log (4).

***Trygger* Verification.** Once edited, the *trygger* can be verified by clicking the "test" button (5). So far, *Trygger* verification comprises: (1) *Trygger* syntax, (2) checking for the triggering ODT to hold a *<select>* tag (this is needed for monitoring).

***Trygger* Creation.** Once the *trygger* is satisfactory verified, the "Create Trygger Alias" link is activated. Besides making the *trygger* locally available, *trygger* creation involves the generation of a *MARS* rule [3]. The so-generated rule is

[11] `https://pubsubhubbub.appspot.com/`

[12] `http://pubsubhubbub.superfeedr.com/`

[13] `http://onekinpush.appspot.com/`

[14] These ODT tables are available for inspection at `https://github.com/yql/yql-tables/blob/master/trygger/JSinterpreter.xml` and .../*RSS-izator.xml*

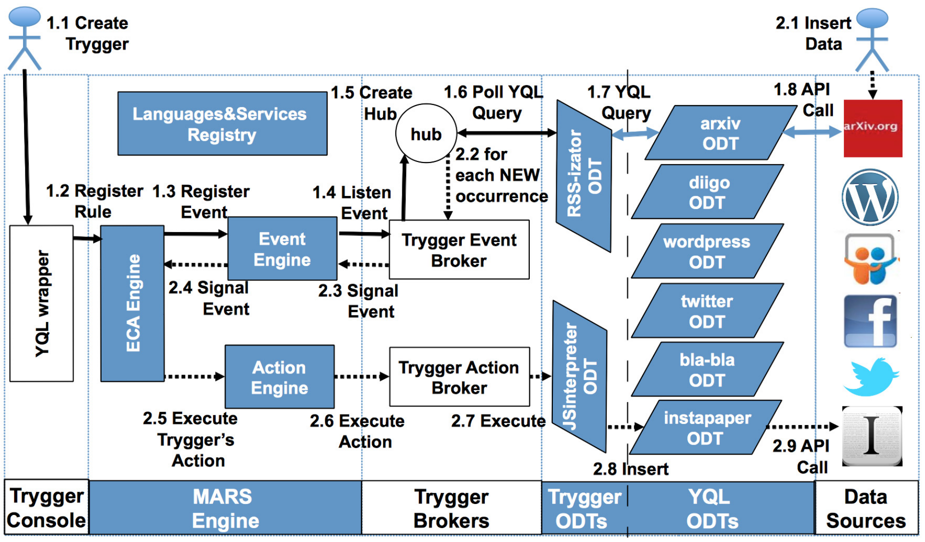


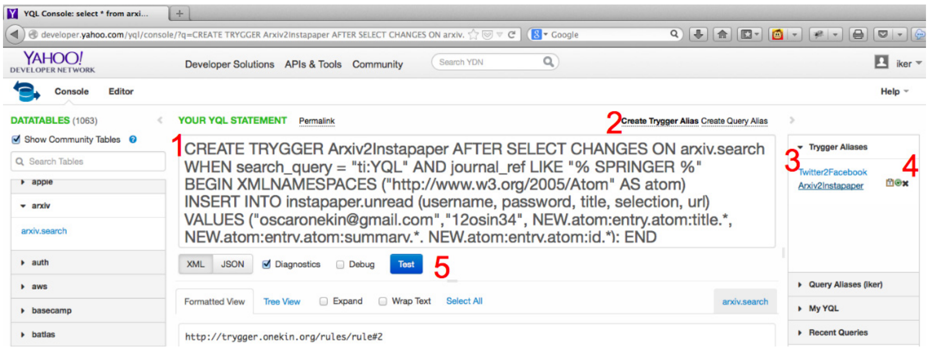
Fig. 5. Interaction diagram among *Trygger* components: *trygger* creation (continuous line) and *trygger* enactment (dotted line) propagating the NEW event occurrence

next registered in MARS (see Figure 5, continuous line). As a result, *MARS* requests the *Event Engine* to register the rule's (1.3) *who* sends the event specification to this listener (1.4). This listener is realized by creating a hub in a *PubSubHubbub* server (1.5). This server periodically polls the triggering website by issuing the *YQL SELECT* query that is obtained from the rule's event (1.6).

***Trygger* Enactment.** The hub detects an incremental change at the next polling time in terms of feeds (2.1 in Figure 5, dotted line). Next, it sends to the *Trygger Event Broker* the updated feed (2.2). The broker creates an event occurrence from the feed, and signals this happening to the *Event Engine* (2.3). The event engine forwards the occurrence to the *ECA Engine* (2.4). The *ECA Engine* retrieves those rules that match the type of new event occurrence, checks their condition and if met, fires the rules in parallel. Rule firing, i.e. enacting rule's action, is conducted through the *Action Engine* (2.5). The *Action Engine* forwards the petition to the domain-specific broker that knows how to handle the statements of the rule's action, i.e. the *Trygger Action Broker* (2.6) which is realized as the *JSinterpreter* ODT (2.7). The *JSinterpreter* processes the *trygger*'s action which can in turn, contain insertion or selection on other ODT tables.

6 Validation

ISO-9126 [5] provides a framework to evaluate quality in use. This section provides a preliminary evaluation of *Trygger* along the ISO-9126's quality-in-use dimensions. We evaluate usability along three aspects:



```
CREATE TRYGGER Arxiv2Instapaper AFTER SELECT CHANGES ON arxiv.search
WHEN search_query = "ti:YQL" AND journal_ref LIKE "% SPRINGER %"
BEGIN XMLNAMESPACES ("http://www.w3.org/2005/Atom" AS atom)
INSERT INTO instapaper.unread (username, password, title, selection, url)
VALUES ("oscaronekin@gmail.com","12osin34", NEW.atom:entry.atom:title.*,
NEW.atom:entrv.atom:summarv.*. NEW.atom:entrv.atom:id.*): END
```

```
http://trygger.onekin.org/rules/rule#2
```

Fig. 6. *YQL* console wrapped for *trygger* management

– Effectiveness, which is the capability of the software product to enable users to achieve specified goals with accuracy and completeness. Indicators of effectiveness include quality of solution and error rates. The *"quality of solution"* is used as the primary indicator of effectiveness, i.e. a measure of the outcome of the user's interaction with the system.
– Productivity, which is the relation between the capability of the software product to enable users to expend appropriate amounts of resources in relation to the effectiveness. Indicators of efficiency include task completion time and learning time. In this study, we use *"task completion time"* as the primary indicator of productivity.
– Satisfaction, which is the users' comfort with and positive attitudes toward the use of the system. Users' satisfaction can be measured by attitude rating scales such as SUMI [11]. In this study, we use *"preference"* as the primary indicator of satisfaction.

6.1 Research Method

Setting. In order to eliminate differences in the perception of *Trygger* due to hardware or bandwidth differences, the study was conducted in a Faculty laboratory. All participants used computers with the same features and a clean installation of Firefox 12.0.

Subjects. The experiment was conducted among 12 graduate students applying in a Master in Web Engineering. They satisfactorily passed a 10 hour course in Web Programmable issues, where they worked with *YQL*. They had accounts in *Twitter* and *Facebook*, and a 92% access social networks on a daily basis while 41% tweet with a weekly frequency. They know about *SlideShare, Arxiv* and *Instapaper* and they were familiarized with the notion of database triggers. Six students knew the existence of *Greasemonkey* but only two had previously installed Greasemonkey scripts.

Procedure. Before starting, a 30' talk with some practical examples of the *Trygger* syntax and the main functions of the *Trygger* console were given.

Initially, the participants installed the *Greasemonkey* extension and the *Trygger* plug-in. Once the framework was deployed, subjects were faced with the creation of the *tryggers* whose outputs correspond to those in Figure 3. In order to measure productivity, participants had to annotate the start time and the finishing time. Finally, the subjects were directed to a *GoogleDocs* questionnaire to gather their opinion about *Trygger*.

Instrument. An online questionnaire served to gather users' experience using *Trygger*. It consisted of four parts, one to collect the participants' background and one for each of the quality-in-use dimensions. In order to evaluate effectiveness, the questionnaire contained the proposed tasks so that participants could indicate if they had performed them and the main problems encountered, while productivity was measured using the minutes taken in such tasks. Satisfaction was measured using 10 questions, using a 5-point Likert scale (1=completely disagree, 5=completely agree).

6.2 Results

Effectiveness. 10 students completed the first *trygger* without additional help. Two had problems with the *Trygger* grammar. This makes us think that the basic *Trygger* syntax is intuitive enough. However, only 6 students completed the second *trygger*. All the participants managed to write the main structure of the *trygger*, but they had problems in dealing with namespaces and correctly identifying the path expressions that obtained the values of the action out of the event occurrence (e.g. *NEW.atom:entry.atom.title.**).

Productivity. Installation of the *Greasemonkey* and the *Trygger* plugs-in took on average 15'. We appreciated a considerable dispersion on the time involved in *trygger* specification. The first *trygger* involved 4' on average while the second took 12' on average. Rationales rest on the same issued previously mentioned: the difficulty in copying with namespaces and Path expressions.

Satisfaction. We evaluated satisfaction for two concerns: the *Trygger* console, and the *Trygger* syntax. According to the results (see Figure 7), the *Trygger* console was in general considered quite supportive. One exception was error messages. Users did not find useful the messages generated by *Trygger* (see next section). As for the *Trygger* syntax, users found easy to equate the *Trygger* syntax to the syntax of triggers in SQL. This "semantic anchoring" might explain the perception of *tryggers* being easy to specify. However, subjects were very critical with the specification of the *trygger*'s action. This is mainly due to the action taken its parameters from NEW. This variable is populated out of an ODT query. This query is based on an ODT table, but the query results are dynamically constructed, i.e. the shape of the output cannot be inferred from the shape of the input ODT. Therefore, users need first to enact the query and look at the output before specifying the *trygger*'s action. This certainly is a main stumbling block.

7 Discussion

Trygger rises different non-functional issues as for usability, extensibility, scalability and reliability. This section discusses these concerns.

Usability (i.e. the ability to easily use *Trygger*). From the aforementioned evaluation, two weaknesses can be identified: the difficulty in describing the *trygger*'s action (the last question of the questionnaire) and the limited information provided by the *Trygger* engine (question 4 of the questionnaire). The former advices from the introduction of enhanced interfaces that traces the *trygger's* enactment, basically, event signaling and action enactment. Debuggers are needed that show the trace of both operations so that users will notice the existence of a mismatch between the returned event occurrence, and the path expressions used to extract the values for the action. This moves us to the second problem: the limitations of the error messages in *Trygger*. So far, the *Trygger* engine does not have its own error messages. Rather, it propagates *YQL* message to the console. But *YQL* is about simple *YQL* statements whereas a *trygger* can be regarded as a compound: a query (the *trygger*'s event) + an update (the *trygger*'s action). This means that a perfectly valid query and a perfectly valid update might deliver a wrong *trygger*. A Trygger-specific error module is needed.

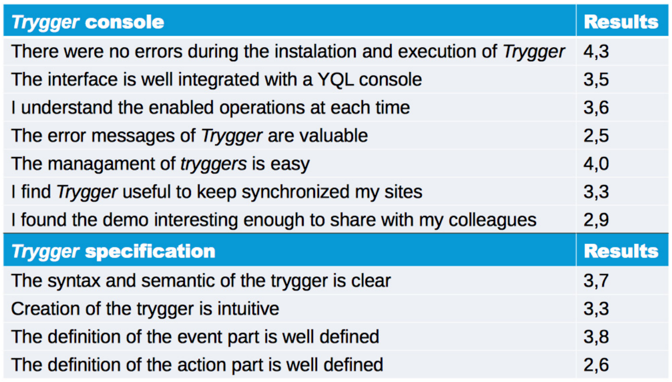
Extensibility (i.e. ability to define *tryggers* no matter the social website). *Trygger* relies upon ODT tables. Yahoo manages an ODT repository which is open to the community[15]. At the time of this writing, above 11054 APIs are documented at *http://www.programmableweb.com/*. Out of this number, 194 have been made *YQL* accessible by the community. We tap into the *YQL* community to keep the ODT repository updated with new comers.

Scalability. *Trygger* processing is highly parallelized. We rely on MARS architecture for this purpose. Besides communication, the polling frequency of the hub also introduces some latency in the process. However, we envision *Trygger* not to be used for time-critical scenarios but rather Web2.0 scenarios where a 5-hour latency between the rising of the event and the enactment of the action is acceptable. Finally, *Trygger* is an agent acting on third-party APIs. Some APIs limit the number of calls per day. Since all calls are made on behalf of *Trygger*, this could be a problem for popular sites. Notice however that popular APIs (e.g. *Twitter*) have a reasonable allowance threshold (around three calls per minute[16]).

Reliability. Reliability is manifold. First, response time. This is not an issue since our scenarios are not time critical. Second, recoverability. We rely on *MARS* for providing storage of rules. In addition, the *Trygger* console keeps a copy of *tryggers* as well. This permits to re-create the rule in *MARS* from the copy kept locally in the *Trygger* installation. Third, safety, specifically about user credentials. We illustrate this scenario for the *Twitter2Facebook trygger* (see Figure 3-

[15] `http://www.datatables.org/`

[16] `https://dev.twitter.com/docs/rate-limiting`



Trygger console	Results
There were no errors during the instalation and execution of *Trygger*	4,3
The interface is well integrated with a YQL console	3,5
I understand the enabled operations at each time	3,6
The error messages of *Trygger* are valuable	2,5
The managament of *tryggers* is easy	4,0
I find *Trygger* useful to keep synchronized my sites	3,3
I found the demo interesting enough to share with my colleagues	2,9
***Trygger* specification**	**Results**
The syntax and semantic of the trygger is clear	3,7
Creation of the trygger is intuitive	3,3
The definition of the event part is well defined	3,8
The definition of the action part is well defined	2,6

Fig. 7. Questionnaire to assess *Trygger* usability. Likert scale is used from Strongly agree (5) to Strongly disagree (1).

A) where *Facebook* credentials were required. Credentials are captured as ODT columns (*i.e. access_token*). This is commonly realized through *OAuth*[17], an open standard for authorization that allows users to share their private resources through API access without having to hand out their credentials, typically username and password.

8 Related Work

Traditionally, cross publishing address two main challenges. First, crossing the border among different media (e.g., print, Web, and TV) [17][16]. Cross media can be defined as any content (news, music, text, and images) published in multiple media. Multiple media indicates that the same content is delivered to end users in more than one medium. The second challenge is to leverage hyperlink-like mechanisms to expand besides HTML pages to other media (i.e. hypermedia). For instance, in [14] the iServer framework is introduced as a generic link management and extensible integration platform that push the boundaries of hyperlinks to paper documents. This work introduces a third challenge akin to the Web 2.0: empowering end-users for defining their own cross publishing strategies. Cross publishing 2.0 admits different variations based on the expressiveness of the strategy (i.e. the criteria for selection, and the associated reactions). *Trygger* provides a push approach to tracking ODT-described resources, and proactively responds by enacting an *YQL*-powered action. Next paragraphs introduce other solutions.

In [12], the focus is on tracking tweets. Rather than using hashtags, the selection criteria to determine tweets of interest is described through an *Resource Description Framework* (*RDF*) query. This implies that tweets need first to be

[17] http://oauth.net/

automatically annotated along a pre-set list of ontologies. The *RDF* query is then addressed along the so-annotated tweets. The reaction is limited to the publications of the tweets meeting the *RDF* query. The architecture also relies on *PuSH* hubs. On the other hand, *ReactiveTags* [9] tracks tagging sites' annotations with specific tags (the so-called reactive tags). Unlike Mendes et al., *ReactiveTags* tracks multiple target sites where the selection and impact criteria are defined semantically in terms of Semantically-Interlinked Online Communities [4] Items. The data is tracked on the target through the API mechanism.

The *Social RSS* [1] *Facebook*'s app links *RSS* sources to a user's *Facebook* account. The app is a feed reader that monitors *RSS* sources and next, publishes each new feed in the user's wall. Reactions are limited to publishing feeds in *Facebook* walls.

Another reactive system for *RSS* tracking is *"Atomate It"* [15]. Both the criteria and the reactions are similar to those of *Trygger*. The difference stems from the architecture. *"Atomate It"* distinguishes two stages. First, *RSS* feeds are loaded. Second, condition-action rules (rather than ECA rules) are run over the previously stored feeds. The tool is aimed at end users so graphical assistance is provided. The downside is expressivity. Complex rules can not be defined, and extending either the *RSS* providers or the action list involves considerable programming. Similarly, *Ifttt*[18] permits users create cross publishing rules using a very intuitive interface, offering a place where users can comment and share their *Ifttt* rules. However, simplicity comes at the expense of customizability. *Ifttt* fixes the way the information is transformed between the two involved services (a.k.a. channels) while also canning the websites that can act as channels.

Which approach is better? Design choices are subject to tradeoffs between factors that will value some attributes while penalizing others. This in turn is influenced by the target audience and the target websites. The work by Mendes et al. focuses on *Twitter* and RDF as main technological platforms. *"Atomate It"* and *Ifttt* are opened to a larger though fixed set of websites. Their efforts favour easy production by trading expressiveness for learnability. *Trygger* explores a different scenario by taping into an existing community (*YQL*). While other approaches depart from raw open APIs (or their RSS counterparts), *Trygger* sits at a higher abstraction layer by starting from ODT tables. Not only does ODT simplifies the development of *Trygger*, it also provides a community that, on expanding *YQL*, is also extending the scope of *Trygger*. In addition, *Trygger* expressiveness attempts to find a compromise by using SQL-like syntax while leaving a backdoor for permitting the use of JavaScript to express more complex reactions.

9 Conclusion

The increasing number of resources kept in the Web together with the growing-up of digital natives make us hypothesize a need for sophisticated cross publishing capabilities. This paper advocates for an ubiquitous, platform-agnostic, do-it-yourself approach to cross publishing. This vision is borne out in *Trygger*, a

[18] `http://ifttt.com`

Firefox plugin that works on top of the *YQL* console. Capitalizing on *YQL*, *Trygger* permits to express cross publishing as *SQL*-like triggers. Next follow-on includes to come up with more elaborate Graphical User Interfaces that open *Trygger* to a wider, less technical, audience.

References

1. Social RSS homepage (2013), `http://apps.facebook.com/social-rss/`
2. Alferes, J., Amador, R., Behrends, E., Fritzen, O., May, W., Schenk, F.: Pre-standardization of the language. Technical report i5-d10. Technical report (2008), `http://www.rewerse.net/deliverables/m48/i5-d10.pdf`
3. Alferes, J., Amador, R., May, W.: A general language for evolution and reactivity in the semantic web. In: Fages, F., Soliman, S. (eds.) PPSWR 2005. LNCS, vol. 3703, pp. 101–115. Springer, Heidelberg (2005)
4. Bojars, U., Breslin, J.G.: SIOC core ontology specification (2007), `http://rdfs.org/sioc/spec/`
5. Davis, I., Vitiello Jr., E.: ISO 9241-11. Ergonomic requirements for office work with visual displays terminals(vdts) (1998)
6. Díaz, O., Arellano, C., Azanza, M.: A language for end-user web augmentation: Caring for producers and consumers alike. ACM Transactions on the Web (TWEB) 7(2), 9 (2013)
7. Fischer, G.: End-user development and meta-design: Foundations for cultures of participation. In: Pipek, V., Rosson, M.B., de Ruyter, B., Wulf, V. (eds.) IS-EUD 2009. LNCS, vol. 5435, pp. 3–14. Springer, Heidelberg (2009)
8. Fitzpatrick, B., Slatkin, B., Atkins, M.: Pubsubhubbub homepage, `http://pubsubhubbub.googlecode.com/`
9. Iturrioz, J., Díaz, O., Azpeitia, I.: Reactive tags: Associating behaviour to prescriptive tags. In: Proceedings of the 22nd ACM Conference on Hypertext and Hypermedia, HT 2011. ACM (2011)
10. Iturrioz, J., Díaz, O., Azpeitia, I.: Generalizing the "like" button: empowering websites with monitoring capabilities. In: ACM (ed.) 29th Symposium On Applied Computing (volume to be published, 2014)
11. Kirakowski, J., Corbett, M.: SUMI: The software usability measurement inventory. Journal of Educational Technology 24(3), 210–212 (1993)
12. Mendes, P.N., Passant, A., Kapanipathi, P., Sheth, A.P.: Linked open social signals. In: Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, Washington, DC, USA, vol. 01, pp. 224–231 (2010)
13. Yahoo! Developer Network. Yahoo query language (YQL) guide (2011), `http://developer.yahoo.com/yql/guide/yql_set.pdf`
14. Norrie, M.C., Signer, B.: Information server for highly-connected cross-media publishing. Information Systems 30(7), 526–542 (2003)
15. VanKleek, M., Moore, B., Karger, D.R., André, P., Schraefel, M.C.: Atomate it! end-user context-sensitive automation using heterogeneous information sources on the web. In: Proceedings of the 19th International Conference on World Wide Web, New York, NY, USA, pp. 951–960 (2010)
16. Veglis, A.: Comparison of alternative channels in cross media publishing. Publishing Research Quarterly 24(2), 111–123 (2008)
17. Veglis, A.A.: Modeling cross media publishing. In: The Third International Conference on Internet and Web Applications and Services, pp. 267–272 (2008)

Ensuring Web Interface Quality through Usability-Based Split Testing

Maximilian Speicher[1,2], Andreas Both[2], and Martin Gaedke[1]

[1] Chemnitz University of Technology, 09111 Chemnitz, Germany
maximilian.speicher@s2013.tu-chemnitz.de,
martin.gaedke@informatik.tu-chemnitz.de
[2] R&D, Unister GmbH, 04109 Leipzig, Germany
{maximilian.speicher,andreas.both}@unister.de

Abstract. Usability is a crucial quality aspect of web applications, as it guarantees customer satisfaction and loyalty. Yet, effective approaches to usability evaluation are only applied at very slow iteration cycles in today's industry. In contrast, conversion-based split testing seems more attractive to e-commerce companies due to its more efficient and easy-to-deploy nature. We introduce *Usability-based Split Testing* as an alternative to the above approaches for ensuring web interface quality, along with a corresponding tool called *WaPPU*. By design, our novel method yields better effectiveness than using conversions at higher efficiency than traditional evaluation methods. To achieve this, we build upon the concept of split testing but leverage user interactions for deriving quantitative metrics of usability. From these interactions, we can also learn models for predicting usability in the absence of explicit user feedback. We have applied our approach in a split test of a real-world search engine interface. Results show that we are able to effectively detect even subtle differences in usability. Moreover, WaPPU can learn usability models of reasonable prediction quality, from which we also derived interaction-based heuristics that can be instantly applied to search engine results pages.

Keywords: Usability, Metrics, Heuristics, Interaction Tracking, Search Engines, Interfaces, Context-Awareness.

1 Introduction

In e-commerce, the usability of a web interface is a crucial factor for ensuring customer satisfaction and loyalty [18]. In fact, Sauro [18] states that "[p]erceptions of usability explain around 1/3 of the changes in customer loyalty." Yet, when it comes to interface evaluation, there is too much emphasis on so-called *conversions* in today's industry. A conversion is, e.g., a submitted registration form or a completed checkout process. While such metrics can be tracked very precisely, they lack information about the actual usability of the involved interface. For example, a checkout process might be completed accidentally due to wrongly labeled buttons. Nielsen [17] even states that a greater number of conversions can be *contradictory* to good usability. In the following, we illustrate this challenge by introducing a typical example scenario.

Scenario. A large e-commerce company runs several successful travel search engines. For continuous optimization, about 10 split tests are carried out per live website and week. That is, slightly different versions of the same interface are deployed online. Then, the one gaining the most conversions is chosen after a predefined test period. The main stakeholder, who studied business administration and founded the company, prefers the usage of *Google Analytics*[1] or similar tools due to their precise and easy-to-understand metrics. Yet, the split testing division would like to gain deeper insights into users' behavior since they know that conversions do not represent usability. Thus, they regularly request more elaborate usability evaluations, such as expert inspections for assessing the interfaces. The stakeholder, however, approves these only for novel websites or major redesigns of an existing one. To him, such methods—although he knows they are highly effective[2]—appear to be overly costly and time-consuming. *Conversion-based split testing* seems more attractive from the company's point of view and is the prime method applied for optimization.

Requirements. The situation just described is a common shortcoming in today's e-commerce industry, which is working at increasingly fast iteration cycles. This leads to many interfaces having a suboptimal usability and potentially deterring novel customers. Thus, we formulate three requirements for a novel usability testing approach that would be feasible for everyday use in industry and support a short time-to-market:

(R1) Effectiveness A novel approach must be more effective than conversion-based split testing w.r.t. determining the usability of an interface.
(R2) Efficiency A novel approach must ensure that evaluations are carried out with minimal effort for both, developers and users. Particularly, deployment and integration must be easier compared to established methods such as expert inspections or controlled lab studies.
(R3) Precision A novel approach must deliver precise yet easy-to-understand metrics to be able to compete with conversion-based split testing. That is, it must be possible to make statements like "Interface A has a usability of 99% and Interface B has a usability of 42%. Thus, 'A' should be preferred."

A solution to the above is to derive usability directly from interactions of real users, such as proposed by [21]. However, they conclude that user intention and even small deviations in the low-level structures of similar webpages influence interactions considerably. This makes it difficult to train an adequate *usability model* M that predicts usability U from interactions $\boldsymbol{I}$ only: $M(\boldsymbol{I}) = U$. Still, the described approach yields great potential.

We consider the pragmatic definition of usability as presented in [20], which is based on ISO 9241-11. Using a corresponding instrument specifically designed

[1] `http://www.google.com/analytics/` (2014-02-01).

[2] For example, [16] state that only five evaluators can find up to 90% of the usability problems in an interface.

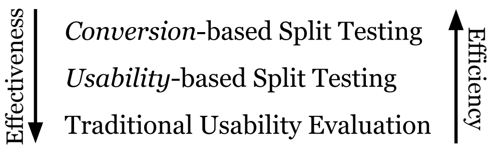

Effectiveness
Conversion-based Split Testing
Usability-based Split Testing
Traditional Usability Evaluation
Efficiency

Fig. 1. Web Interface Usability Evaluation: the competing approaches (rough overview)

for correlation with client-side interactions [20], we propose a general approach to *Usability-based Split Testing* rather than considering conversions. To achieve this, we provide *WaPPU*—a tool that caters for (a) user interaction tracking, (b) collecting usability judgments from real users, (c) training usability models and (d) correlation of the obtained data. By design, the concept of Usability-based Split Testing enables developers to *ensure the quality of a web application* w.r.t. its interface usability at higher effectiveness than conversion-based split testing and higher efficiency than traditional approaches to usability evaluation (Fig. 1).

Making use of WaPPU ("*Wa*s that *P*age *P*leasant to *U*se?") we performed a usability-based split test of a real-world search engine results page (SERP). We paid specific attention to user intention and differences in low-level page structure to overcome the problems pointed out in [21]. From the study results, we derived interaction-based usability models and quantitative heuristic rules for SERPs. These can be instantly applied to user interactions collected on a SERP for a reasonable approximation of usability at very high efficiency.

In the following section, we give an overview of related work and describe the initial user study motivating our novel approach. Subsequently, we explain the concept of Usability-based Split Testing (Sec. 3) and the corresponding tool WaPPU (Sec. 4). The evaluation involving two web interfaces of a real-world search engine are presented in Section 5, followed by our findings (Sec. 6). Current limitations of our approach and potential future work are discussed in Section 7 before giving concluding remarks in Section 8.

2 Related Work

Our research is related to a wide variety of existing work. In particular, we are going to refer to *automatic* and *metrics-based* approaches to usability evaluation that are partly based on *user interaction analysis*. We also present an earlier study on the feasibility of quantitative interaction-based usability evaluation.

2.1 Automatic Approaches to Usability Evaluation

User Interaction Analysis. Atterer et al. [1] present a tool for client-side user interaction tracking. After having collected information about cursor behavior, keyboard strokes or dwell time, one can use these events to visualize a users interactions on a webpage. From these, the authors aim to infer implicit interactions, such as hesitation before filling in an input field [1]. This is a useful tool for facilitating more automatic usability tests and provides developers with valuable information. *m-pathy*[3] is a commercial tool for qualitative user behavior inspection that follows the concept described by [1]. The tool features additional metrics that are, however, in analogy to conversion-based split testing, e.g., the number of checkout processes and similar.

Web Usability Probe [2] is a more sophisticated tool also allowing for automatic remote usability testing. It is possible to define optimal logs for given tasks, which are then compared to the client-side logs actually produced by the user. De Vasconcelos and Baldochi Jr. [4] follow a similar approach that compares users' interactions against pre-defined patterns.

In contrast to our novel approach, all of the above methods—although as well aiming at usability improvement—have different focuses. None derives *quantitative* statements about usability from the observed interactions, which would enable direct comparison of interfaces. Rather, interpretation of the delivered qualitative information is largely up to a developer or dedicated usability evaluator.

Navalpakkam and Churchill [12] investigate the possibility to infer the user experience of a web interface from mouse movements. In a user study, they find that certain features of interaction (e.g., hovers, arrival time at an element) can be used to predict reading experience and user distraction with reasonable accuracy. Yet, they investigate only these specific aspects. Particularly, the authors do not focus on providing interaction-based measures of usability or user experience for quantitative comparison of interfaces.

Website Checking. Tools such as *AChecker* [6] and *NAUTICUS* [3] aim at automatic checking of websites according to certain criteria and heuristics. While the first specifically focuses on web accessibility, the second tool also takes into account usability improvements for visually impaired users. Both tools are particularly able to automatically suggest improvements regarding the investigated interfaces. Yet, they only consider static criteria concerned with structure and content of a website rather than actual users' interactions.

A/B Testing. *AttrakDiff*[4] is a tool that enables A/B testing of e-commerce products for user experience optimization. That is, based on a dedicated instrument, the hedonic as well as pragmatic quality of the products are compared [11]. While this may seem very similar to our proposed approach, it has to be noted that the aim of AttrakDiff is different from Usability-based Split Testing. Particularly, the tool leverages questionnaire-based remote asynchronous evaluation rather than focusing on user interactions. Also, qualitative, two-dimensional statements about user experience are derived, which has to be clearly distinguished from usability and quantitative metrics thereof.

[3] `http://www.m-pathy.com/cms/` (2014-02-24).
[4] `http://attrakdiff.de/` (2014-02-24).

2.2 Metrics-Based Approaches to Usability Evaluation

Contrary to the above approaches, Nebeling et al. [14] take a step into the direction of providing quantitative metrics for webpage evaluation. Their tool analyzes a set of spatial and layout aspects, such as *small text ratio* or *media–content ratio*. These metrics are static (i.e., purely based on the structure of the HTML document) and specifically aimed at large-screen contexts. In contrast, our goal is to provide *usability-in-use* metrics based on users' dynamic interactions with the webpage.

W3Touch [15] is a metrics-based approach to adaptation of webpages for touch devices. This means certain metrics of a website, e.g., *average zoom*, are determined from user interactions on a per-component basis. Components with values above a certain threshold are then assumed to be "critical" and adapted according to rules defined by the developer. This is a very promising approach that is, however, specifically aimed at touch devices. Moreover, the webpage metrics that identify potentially critical parts of a webpage are not transferred into more precise statements about usability.

2.3 Motivating Study

In the following, we address earlier work of the authors of this paper [21] that motivates the concept of Usability-based Split Testing.

In [21], we have tried to solve the already described conflict between traditional usability evaluations and conversion-based split testing by learning usability models from user interactions. For this, we have collected user interaction data on four similarly structured online news articles. Study participants had to answer a specific question about their assigned article. However, only two of the articles contained an appropriate answer. Once they found the answer or were absolutely sure the article did not contain it, participants had to indicate they finished their task and rate the web interface of the article based on the novel Inuit usability instrument [20]. Inuit contains seven items designed for meaningful correlation with client-side interactions (e.g., , "cursor speed positively correlates with confusion") from which an overall usability score can be derived.

All articles featured a single text column with a larger image on top and a sidebar which contained additional information and/or hyperlinks. Two articles featured a short text ($\sim$1 page) while the remaining two featured a longer text ($\geq$2 pages). Moreover, two of the articles featured images within or close to the text column. Still, the high-level structures of the articles were similar. The lower-level differences were chosen by purpose to provoke differences in user behavior and usability judgments. Also, by providing an answer to the participant's task in only two of the four articles, we have simulated different user intentions according to [8]. That is, participants who could find an answer acted like *fact finders* while the remaining participants behaved like *information gatherers*.

We used a dedicated jQuery plug-in to track a set of well-defined interactions (e.g., clicks, hovers, scrolling distance etc.). These interactions were recorded separately for: (1) the whole page; (2) a manually annotated area of interest, i.e., the article text; (3) all text elements; (4) all media elements; (5) text elements within the area of interest; and (6) media elements within the area of interest.

To give a representative example, the interaction feature $hoverTime_{text}$ describes the aggregated time the user spent hovering text elements anywhere on the page. Furthermore, all interaction feature values were normalized using appropriate page-specific measures. For example, the page dwell time was divided by the length of the article text to ensure comparability across the four webpages.

The main hypothesis investigated in the study was *whether it is possible to learn a common usability model from similarly structured webpages.* Such a model should be able to also predict the usability of a webpage which did not contribute training data, as long as its structure is similar to a certain degree. However, when we correlated the collected interactions and usability judgments, we found huge differences between the four news articles. In fact, there was no interaction feature that showed a considerable correlation with usability for all four investigated webpages.

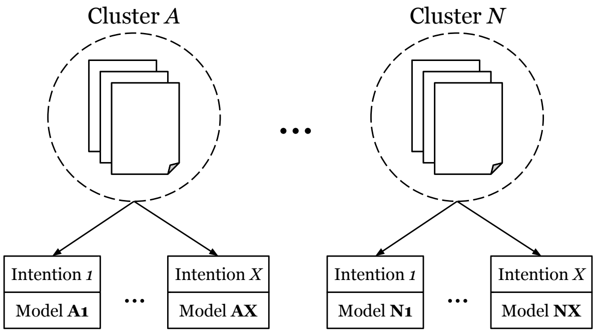


Fig. 2. Concept of a general framework for providing interaction-based usability models

This result indicates that user behavior depends on low-level webpage structure and intention more strongly than assumed, i.e., *interactions* are a function of *usability*, *structure* and *intention*.

Thus, we concluded that a general framework for interaction-based usability evaluation requires additional preprocessing steps: (1) structure-based clustering of webpages; (2) determining user intention, e.g., following the approach proposed by [8]; and (3) providing a common usability model per cluster and intention.

That is, for X types of user intention, a corresponding framework would have to provide X usability models per webpage cluster (Fig. 2). For example, assume a cluster contains all blogs using the same WordPress template. Then one would have to train different models for users *just browsing around* and users looking for a *certain piece of information* since these two intentions cause considerable differences in behavior. In the remainder of this paper, we address how to derive appropriate usability models and heuristics w.r.t. the requirements just described.

3 Usability-Based Split Testing

We propose *Usability-based Split Testing*, which is a feasible trade-off between effectiveness and efficiency, as an alternative to established approaches (Fig. 1). That is, we aim at significantly better predictions of usability than can be done using conversions. Besides, we want to be more efficient than established methods of usability evaluation. To achieve this, we have designed a two-step process:

1. Track user behavior on the interfaces of a split test—i.e., the *interfaces-under-test*—and apply the resulting interaction metrics to **heuristic rules** for usability evaluation, e.g., "a higher cursor speed indicates more confusion". Test whether the difference between the interfaces is *significant.*
2. If the result is not significant, more specific information is required. Thus, add a *usability questionnaire* to one of the interfaces. From the answers and the tracked interactions, learn more specific **usability models** that can be applied to the remaining interfaces for predictions of usability.

For realizing these steps and meeting requirements **(R1)**–**(R3)**, as described in Section 1, our novel approach follows a set of well-defined principles that will be introduced in the following.

3.1 Component-Based Interaction Tracking

The major goal of our approach is to overcome the problems regarding interactions and low-level page structure, as described in Section 2.3. During the study motivating this paper, interaction feature values were calculated on a very fine-grained basis. Particularly, interactions on any text or media element were considered for analysis, no matter how tiny or unimportant. This means that removing some minor elements from a webpage—such as text snippets in a sidebar—would already impact the values of interaction features. Also, only normalized absolute values were considered, rather than paying attention to relative distributions of interactions across the webpage.

To address this issue of interactions being highly dependent on low-level structure, we follow a *component-based approach.* For this, an interface-under-test, i.e., a single webpage, is divided into higher-level components such as the whole navigation bar rather than considering individual links. This approach partly follows the concepts of areas of interest [7] and critical components [15]. The rest of the webpage is treated as a separate, remaining component while the lower-level structure within a component is considered a *black box.* It is also possible to apply this to components in the context of other approaches, e.g., the WebComposition approach [5]. Since we intend to track interactions on a per-component basis, in this way small changes to the lower-level structure—e.g., removing minor text snippets—do not have an impact on feature values. *Usability models learned from such component-based interactions can then be applied to different webpages as long as the large-scale structure remains the same.*

3.2 Interaction-Based Heuristic Rules for Usability Evaluation

Interactions tracked in the context of a usability-based split test can be easily applied to pre-defined heuristic rules. To give just one example, assume a rule stating that *higher cursor speed positively correlates with user confusion.* Then, if the users of one interface-under-test produce significantly lower cursor speeds than users of another, this is a clear indicator of less confusion. By design, this variant of our approach is as efficient as conversion-based split testing *(R2).* That is, it can be very quickly deployed on online webpages and does not bother the user with requests for explicit feedback. Moreover, the collected interaction-based metrics are precise and easily interpretable using the given heuristic rules *(R3).*

A drawback of this variant is the fact that the rules used need to be determined in a different setting of the same context (i.e., similar high-level structure, similar user intention) first. That is, a dedicated training phase is required, e.g., a controlled user study during which explicit usability judgments are correlated with interactions. Since the applied heuristic rules originate from a different setting, they cannot be a perfect measure of usability for the interfaces-under-test. Rather, they can only give reasonable approximations, but still provide more insights into users' actual behavior than conversions *(R1).* However, if this approach fails to deliver significant results, one needs to obtain more precise information for predicting usability by leveraging corresponding models.

3.3 Leveraging Usability Models

The second variant of our Usability-based Split Testing approach uses models for predicting usability. For this, one interface-under-test is chosen to deliver training data. That is, users of the interface are presented with a questionnaire asking for explicit, quantitative judgments of usability. This questionnaire is based on Inuit [20], an instrument describing usability by a set of only seven items: *informativeness*, *understandability*, *confusion*, *distraction*, *readability*, *information density* and *accessibility.* In this way, the number of questions a user has to answer is kept to a minimum [20]. The items have also been specifically designed for meaningful correlation with client-side user behavior [20]. Together with the collected interactions, explicit judgments are then used for training appropriate models based on existing machine learning classifiers. Since the interfaces-under-test all feature the same high-level structure—in accordance with *component-based interaction tracking*—these models can be applied to the interactions of the remaining interfaces for predictions of usability.

This variant of our approach cannot reach the same efficiency as conversion-based split testing since parts of the users are faced with requests for explicit feedback. Yet, by design, it is more efficient than traditional methods such as remote asynchronous user studies *(R2).* Given the minimum of two interfaces-under-test, only 50% of our users are presented with questionnaires, compared to 100% of the participants in a controlled user study. Moreover, our approach can be easily applied to online web interfaces. It does not require a cumbersome study set-up since we rely on interactions and judgments of real users only.

In comparison to conversions or heuristic rules, models provide considerably more precise insights into users' behavior and its connection to usability *(R1)*. Also, questionnaires and models deliver an easily interpretable set of quantitative metrics in terms different usability aspects (*informativeness*, *understandability* etc.) for comparing interface performance *(R3)*.

4 The WaPPU Tool

To provide a ready-to-use framework for Usability-based Split Testing, we have designed a novel context-aware tool called *WaPPU*.The tool caters for the whole process from interaction tracking to deriving correlations and learning usability models. Based on the principles of Usability-based Split Testing, we have implemented WaPPU in terms of a central split testing service. This service has been realized using *node.js*[5]. Split testing projects are created in the WaPPU dashboard (Fig. 3), which provides the developer with ready-to-use JavaScript snippets that simply have to be included in the different interfaces-under-test. The only other thing required for deployment of the split test is a client-side jQuery plug-in for component-based interaction tracking. The overall architecture of WaPPU can be seen in Figure 3. The current implementation supports at most two interfaces per split test, i.e., only A/B testing is supported.

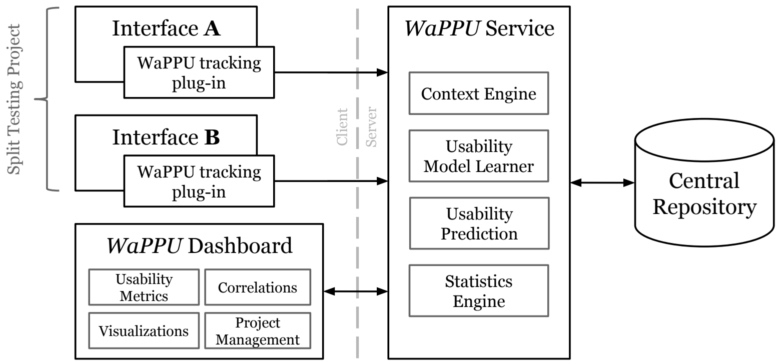


Fig. 3. Architecture of WaPPU

Interaction Tracking. Our tool tracks a total of 27 well-defined user interaction features, of which the most expressive ones are shown in Table 1. They have been derived from existing research [7,12,15,19] as well as video analyses of real search engine users. The features are tracked for each component defined by the developer, except for features annotated with an asterisk, which cannot be applied to individual components. Moreover, each feature is tracked for the

[5] `http://nodejs.org/` (2014-02-21).

Table 1. Selection of interaction features tracked by WaPPU (* = whole page feature only). The complete list can be found in our online appendix [22].

label	description	source
charsTyped	# characters typed	
cursorMoveTime	time the mouse cursor spends moving	[19]
cursorSpeed	*cursorTrail* divided by *cursorMoveTime*	[7,19]
cursorSpeedX	cursor speed in X direction	[7]
cursorStops	# cursor stops	[7]
cursorTrail	length of cursor trail	[7,19]
hovers	# hovers	[19]
hoverTime	total time spent hovering the component	[19]
pageDwellTime*	time elapsed between loading and leaving the page	[7]
scrollDirChanges*	# changes in scrolling direction	[15]
scrollMaxY*	maximum scrolling distance from top	[7]
scrollPixelAmount*	total amount of scrolling (in pixels)	[7]
textSelections	# text selections	
textSelectionLength	total length of all text selections	

whole web interface, which gives an additional implicit component. This gives us the chance to derive the relative distribution of features across the page, e.g., "25% of the total cursor trail lie in the navigation component". If a developer defines x components in their web interface and specifies that all features shall be considered, WaPPU tracks a total of $20(x + 1) + 7$ features during the split test (7 features are applied to the whole page instead of components).

Usability Judgments. WaPPU offers the option to automatically show a questionnaire when users leave an interface-under-test, in case they have agreed to contribute training data. This questionnaire contains the seven usability items of Inuit [20], each formulated as a question and to be answered based on a 3-point Likert scale. Since the value of an item is thus either -1, 0 or $+1$, we get an overall usability value that lies between -7 and $+7$. These values are what we refer to as *quantitative measures/metrics of (aspects of) usability* in the remainder of this paper. The questionnaire can be shown on either none, one or all of the interfaces-under-test in a split test. If no interface features a questionnaire, the functionality of WaPPU is reduced to collecting interactions only, i.e., for use with *usability heuristics* (cf. Sec. 3.2).

If it is featured on one interface, WaPPU automatically learns seven models—one per usability item—based on the users' answers. These models are associated with the corresponding split testing project and stored in WaPPU's central repository (Fig. 3). They are automatically applied to the remaining interfaces for *model-based usability prediction* (cf. Sec. 3.3). The current implementation of

WaPPU uses the updateable version of the Naïve Bayes classifier[6] provided by the WEKA API [10].

Finally, in case all interfaces feature the questionnaire, the developer receives the most precise data possible. This case requires no models and is particularly useful for *remote asynchronous user studies* from which one can derive heuristic rules for usability evaluation (cf. Sec. 3.2). It is not intended for evaluation of online interfaces since the amount of questionnaires shown to real users should be minimized.

Context-Awareness. The context of a user is automatically determined by WaPPU and all collected interactions and usability judgments are annotated accordingly. In this way, it is possible to integrate context into a usability model since different contexts trigger different user behaviors. Currently, we consider two aspects that to a high degree influence a user's interactions: *ad blockers* and *screen size*. That is, the context determined by our tool is a tuple $(\mathit{adBlock}, \mathit{screenSize})$ with $\mathit{adBlock} \in \{\mathit{true}, \mathit{false}\}$ and $\mathit{screenSize} \in \{\mathit{small}, \mathit{standard}, \mathit{HD}, \mathit{fullHD}\}$. For this, we refer to the most common screen sizes and define: $\mathit{small} < 1024 \times 768 \leq \mathit{standard} < 1360 \times 768 \leq \mathit{HD} < 1920 \times 1080 \leq \mathit{fullHD}$.[7]

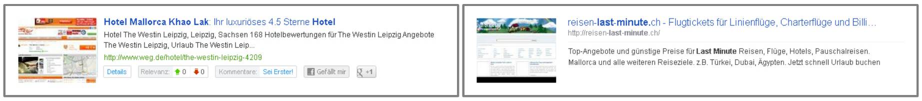
Small-screen and touch devices are not supported in the current version of WaPPU. They are detected using the MobileESP library[8] and corresponding data are ignored.

5 Evaluation

We have engaged the novel concept of Usability-based Split Testing for evaluating the interface of a real-world web search, which is currently a closed beta version and developed by the R&D department of *Unister GmbH*. For this, we have redesigned the standard search engine results page (SERP) and put both the old and redesigned versions of the interface into an A/B test using WaPPU. According to component-based interaction tracking as one principle of Usability-based Split Testing (cf. Sec. 3.1), we have defined two high-level components within the SERPs: the container element containing all search results (`#serpResults`) and the container element containing the search box (`#searchForm`). From this split test, we have obtained (a) an evaluation of the two SERPs, (b) corresponding usability models and (c) a set of interaction-based heuristics for general use with SERPs of the same high-level structure. Results suggest that our approach can effectively detect differences in interface usability and train models of reasonable quality despite a rather limited set of data.

The following describes the research method for evaluating the approach, the concrete test scenario, and presents the evaluation results. Datasets for reproducing results and detailed figures can be found in our online appendix [22].

[6] `http://weka.sourceforge.net/doc.dev/weka/classifiers/bayes/NaiveBayesUpdateable.html` (2013-10-07).
[7] Cf. `http://en.wikipedia.org/wiki/Display_resolution` (2014-02-12).
[8] `http://blog.mobileesp.com/` (2014-02-12).



Hotel Mallorca Khao Lak: Ihr luxuriöses 4.5 Sterne **Hotel**
Hotel The Westin Leipzig, Leipzig, Sachsen 168 Hotelbewertungen für The Westin Leipzig Angebote The Westin Leipzig, Urlaub The Westin Leip...
http://www.weg.de/hotel/the-westin-leipzig-4209

reisen-**last-minute**.ch - Flugtickets für Linienflüge, Charterflüge und Billi...
http://reisen-**last-minute**.ch/
Top-Angebote und günstige Preise für **Last Minute** Reisen, Flüge, Hotels, Pauschalreisen. Mallorca und alle weiteren Reiseziele. z.B. Türkei, Dubai, Ägypten. Jetzt schnell Urlaub buchen

Fig. 4. Search result from the original SERP (left) vs. search result from the novel SERP redesigned by three usability experts (right)

5.1 Method

The evaluation was carried out as a remote asynchronous user study whose workflow oriented at [13]. Participants were recruited via internal mailing lists of the cooperating company. Since user intention considerably affects interactions (cf. Sec. 2.3), we intended to minimze fluctuations in this respect. Thus, we definined a *semi-structured* task to simulate that all participants act according to a common intention, i.e., "Find a birthday present for a good friend that does not cost more than 50 Euros." We assumed that the vast majority of users would not immediately have an adequate present in mind and thus behave like *information gatherers* [8]. Additionally, in order to reduce context to different screen sizes only, participants were instructed to disable any ad blockers.

Each participant was randomly presented with one of the two SERP interfaces for completing their task. Before leaving a results page, they had to rate its usability using the INUIT questionnaire displayed by WaPPU. That is, we used a configuration of WaPPU which triggers a questionnaire in both interfaces in the A/B test. Since a user might trigger several searches and thus view several different SERPs, they potentially had to answer the questionnaire more than one time during the study. This means that one study participant could produce several datasets, each containing interactions and usability judgments. Answering one questionnaire per results page is necessary since different searches lead to different results, which influences usability items such as *informativeness* and *information density*. Participants were instructed to click a "finish study" button, once they found an adequate present. Clicking the button redirected to a final questionnaire asking for demographic information.

5.2 Interface Redesign

The web search's standard SERP interface was redesigned by three experts in order to increase its usability. The redesign was carried out according to established guidelines as well as the experts' own experiences with interface design and usability evaluations. One of the experts was a graphic designer and one was an interaction designer holding an according Master's degree, both with several years of experience in industry. The third expert was a PhD student with focus on human-computer interaction.

Some representative points concerning the redesign were: (a) better visual separation of results, (b) more whitespace, (c) giving text more space in terms of a greater line height, (d) aligning text and image elements more consistently,

(e) removing unnecessary comment and social media buttons and (f) reducing the amount of advertisements. Although the changes were rather subtle, we particularly assumed less *confusion* and *distraction* as well as better *readability* and *information density*. A visual comparison of exemplary search results from the two interfaces can be found in Fig. 4.

5.3 Results

We recruited 81 unique participants who contributed 198 individual datasets, i.e., they triggered 198 searches/SERPs. 17 of the participants were familiar with the investigated web search (i.e., they had used it before); 37 answered the final questionnaire (23 male). In general, participants stated they privately surf the internet for 2–3 hours per day, mostly for social networking (N=23) and reading news (N=22). The mostly used search engine is Google (N=35), in general several times a day (N=34) and usually for knowledge (N=31) and product search (N=21). On average, participants were 31.08 years old (σ=5.28).

During the study, we registered two different contexts: *HD* (N=46) and *full HD* (N=35). One participant was excluded from the analysis, because they delivered invalid data. For our evaluation, we additonally distinguish between users who were *not familiar* and users who were *familiar* with the web search since they produced considerably different results.

Interface Evaluation. First, we have a look at the usability evaluations of the two SERP interfaces w.r.t. the questionnaires filled out by the participants. That is, we investigate whether our approach was able to detect the difference in usability originating from the experts' interface redesign. For this, we have carried out four analyses regarding the two contexts *HD* and *full HD* as well as familiarity with the investigated web search.

The largest amount of datasets (89) was produced by users with HD screens who were not familiar with the web search. Participants also not familiar with the interface, but using full HD screens contributed 52 datasets. This makes a total of 141 datasets from novel users. In contrast, participants who were familiar with the web search contributed 47 datasets (30 HD, 17 full HD).

As is apparent from Table 2, the largest group of users (*HD/not familiar*) found the redesigned SERP to be significantly[9] better regarding the aggregated usability (i.e., the sum of all individual items; μ=2.45, σ=2.46). Moreover, the new interface performed significantly better concerning *distraction* (μ=0.62, σ=0.62) and *information density* (μ=0.43, σ=0.67). This matches our assumptions based on the experts' redesign. In general, the new SERP performs better regarding all usability items, although not always statistically significant.

In contrast, analysis of HD screen users familiar with the web search did not show a significant overall difference between the two SERPs. Yet, they judged the old interface to be significantly better concerning the individual item *confusion* (μ=0.69, σ=0.48). On average, it was also judged to be less *distracting*

[9] All tests of significance were carried out using the *Mann–Whitney U test* (α=0.05), which is particularly suitable for non-normally distributed independent samples.

Table 2. Evaluations by participants not familiar with the web search who used an HD screen (A = old interface, B = new interface)

	A (N=47)		B (N=42)		
usability item	μ	σ	μ	σ	significance
informativeness	-0.17	0.84	-0.02	0.84	—
understandability	0.34	0.70	0.45	0.67	—
confusion	0.30	0.78	0.38	0.70	—
distraction	0.36	0.74	0.62	0.62	$p<0.05$, W=798.5
readability	0.45	0.65	0.52	0.71	—
information density	0.04	0.69	0.43	0.67	$p<0.01$, W=692
accessibility	0.06	0.67	0.07	0.75	—
usability	1.38	2.96	2.45	2.46	$p<0.05$, W=782

(μ=0.85, σ=0.38) and have better *readability* (μ=0.62, σ=0.51) and *accessibility* (μ=0.38, σ=0.65). This finding is contrary to our assumptions. Rather, it indicates that users get accustomed to suboptimal interfaces and seem to be confused by changes even if they yield better usability from a more objective point of view.

Concerning the context *full HD/not familiar*, our analysis shows no significant differences between the two interfaces. However, results suggest that the usability of the old interface is better on average. Contrary, the redesigned SERP on average indicates better performance regarding *information density* (μ=0.19, σ=0.83) and *confusion* (μ=0.06, σ=0.85). Finally, full HD users who were familiar with the web search saw the biggest difference between the two interfaces. They judged the new SERP to be significantly better concerning *distraction* (μ=0.80, σ=0.42), *readability* (μ=0.60, σ=0.52), *information density* (μ=0.10, σ=0.57), *accessibility* (μ=0.50, σ=0.71) and aggregated usability (μ=2.60, σ=2.41). However, this context contained the smallest number of datasets (17) and therefore cannot be considered to be representative.

Usability Models. Based on the most representative dataset (Tab. 2) we have trained and tested Random Forest[10] classifiers for predicting usability across interfaces. This is in analogy to WaPPU's functionality of providing the questionnaire only in one interface and guessing the usability of the second interface from automatically learned models. Particularly, we intend to investigate whether component-based interaction tracking (cf. Sec. 3.1) is feasible for predicting the usability of a different webpage that did not contribute training data. For this, we take interaction data and usability judgments from the old SERP and train models from these—one for each Inuit usability item. The interaction data from the redesigned SERP are used as the test set for these models.

In a first step, we have selected the most expressive interaction features for each model. This has been done using *Correlation-based Feature Subset Selection*

[10] `http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/RandomForest.html` (2014-03-23).

[9] in combination with best-first search. That is, we have selected "[s]ubsets of features that are highly correlated with the class [to be predicted] while having low intercorrelation"[11]. Both functions are provided by the WEKA API [10]. Subsequently, we have trained the models based on the selected features and used our test set to evaluate them.

In general, the quality of the trained models was reasonably good. We obtained the most precise predictions for the item *distraction* (F-measure = 0.518), which was also one of the significant items for the considered context. In contrast, the item *readability* yielded the least precise predictions (F-measure = 0.296). The amount of training and test data was rather small for the investigated context (47 and 42 data sets, respectively). Thus, we assume better prediction quality with a larger amount of real-world users since correlations would then become more homogeneous, as has been observed in [19]. The precise results of the model evaluation can be found in our online appendix [22].

6 Key Findings of the User Study

The results from the largest and most representative group of participants *HD/not familiar* (Tab. 2) confirm that our approach is able to effectively detect differences in the usability of two versions of the same interface. We moreover found that users get used to interfaces and thus become less receptive to adjustments, even if these aim at better usability. What remains to be investigated are the differences between judgments from HD and full HD users. In particular, it requires deeper insights into users' actual behavior to understand why users familiar with the investigated web search produced very contradictory evaluations when differentiating between screen resolutions.

The usability models trained from our data underpin that the component-based tracking approach can reduce variations in users' interactions, which are caused by differences in lower-level structure. This was a major problem during the motivating study (cf. Sec. 2.3). Results suggest that WaPPU is able to predict interface usability based on adequate models with reasonable effectiveness.

Based on the feature selection process for learning usability models and Pearson's correlations r, we have additionally derived heuristic rules for SERPs, which are summarized in Figure 5. Regarding the dataset these rules are based on, their validity is theoretically restricted to *HD screen* users. Still, they can be applied to any SERP—as long as it is of similar structure and has the same components defined as the SERPs investigated in our evaluation—since many of the included features (e.g., *page dwell time*) do not strongly depend on screen resolution. To give just one example, a developer could monitor interactions on two SERPs. If *page dwell time* and *maximum scrolling distance* are significantly lower on one SERP, this is a clear signal for better *information density*. Yet, results must be interpreted with caution, as we have only investigated the user type *information gatherer* in our study. If it is not possible to obtain significant results

[11] `http://weka.sourceforge.net/doc.dev/weka/attributeSelection/CfsSubsetEval.html` (2014-02-20).

- Better **informativeness** is indicated by
  - a lower absolute *cursor speed* on the search box (r=-0.21);
  - a higher relative amount of *hovers* on the search results (r=0.40).
- Better **understandability** is indicated by
  - a lower absolute *cursor speed* on the search box (r=-0.46);
  - a higher relative amount of *hovers* on the search results (r=0.24).
- Less **confusion** is indicated by
  - a lower relative *cursor speed (X axis)* on the search box (r=-0.49);
  - a lower absolute *maximum scrolling distance from top* (r=-0.44);
  - a lower absolute *amount of scrolling (in pixels)* (r=-0.33).
- Less **distraction** is indicated by
  - a lower absolute amount of *cursor stops* (r=-0.26);
  - a smaller absolute *length of the cursor trail* (r=-0.25).
- Better **readability** is indicated by
  - a lower absolute *page dwell time* (r=-0.21);
  - a smaller absolute amount of *text selections* (r=-0.27);
  - a smaller absolute *length of text selections* (r=-0.39).
- Better **information density** is indicated by
  - a lower absolute *page dwell time* on the search box (r=-0.11);
  - a lower absolute *maximum scrolling distance from top* (r=-0.27).
- Better **accessibility** is indicated by
  - a lower absolute amount of *characters typed* into the search box (r=-0.27);
  - a lower absolute amount of *changes in scrolling direction* (r=-0.31).

Fig. 5. Heuristic rules for usability evaluation of SERPs, as derived from our user study

from the heuristics, one must switch to a more effective method—e.g., leveraging specifically trained models, as described earlier.

7 Limitations and Future Work

We are aware of the fact that usability is a hard-to-grasp concept that is difficult to measure in an objective manner—if possible at all. However, our approach is able to yield reasonable approximations of usability in a quantitative and easy-to-understand form. This is particularly valuable in today's IT industry with its short time-to-market. If existing conversion-based analyses are augmented with Usability-based Split Testing, it will be possible to detect major shortcomings in web interfaces without having to carry out costly and/or time-consuming evaluations (yet, our approach only detects differences between interfaces-under-test and does not directly drive adequate changes for better usability). If results delivered by our method are not significant, it is still possible to apply such evaluations, which are more effective yet less efficient. However, we intend to minimize the need for the latter.

As has been pointed out earlier (cf. Sec. 2.3), a user's intention has considerable impact on their behavior. However, we have not yet considered intention as a factor in the design of our Usability-based Split Testing tool WaPPU. Rather, we modeled the user study for our evaluation in such a way that all users had to behave the same. Currently, we are investigating how intention can be derived from the interaction features tracked by WaPPU. According to [8], features such as the *page dwell time* can indicate user behavior. In future versions of WaPPU, we intend to add an extra question to the questionnaire asking for the user's intention. In this way, we can train an additional model for determining intention before applying adequate usability models or heuristics.

The current version of WaPPU is restricted to processing mouse and keyboard input. Yet, small-screen touch devices are gaining more and more popularity. Therefore, a major part of our future work will be to transfer Usability-based Split Testing into the context of touch devices. It will be particularly interesting to investigate how the different set of interaction features (e.g., missing *cursor trail*, new *zooming interaction*) affects usability prediction quality. First steps into this direction have already been taken by Nebeling et al. [15].

Finally, we intend to integrate our approach into the WebComposition process model [5] for enabling continuous evaluation of evolving widget-based interfaces.

8 Conclusions

This paper has presented *Usability-based Split Testing*—a novel method for *ensuring web inferface quality* based on quantitative metrics and user interactions. We have also introduced a corresponding A/B testing tool called *WaPPU*. Our approach intends to determine the usability of an interface more effectively than conversion-based methods while being more efficient than traditional approaches like expert inspections or controlled lab studies. To realize this, our method determines usability based on users' interactions. That is, we track interactions and apply them to either pre-defined heuristic rules or models trained with the help of users who answered an additional questionnaire. In this way, we obtain quantitative approximations of usability for empirically comparing web interfaces.

In a user study with 81 participants, we have applied our approach to the standard version and a redesigned version of a real-world SERP. Results show that our tool is able to detect the predicted differences in usability at a statistically significant level. Moreover, we were able to train usability models with reasonable prediction quality. Additionally, a set of key usability heuristics for SERPs could be derived based on user interactions. The study findings underpin the feasibility of the proposed approach.

Future work includes transferring the approach into the context of touch devices. Moreover, future versions of WaPPU shall be able to determine a user's intention before selecting appropriate usability heuristics and models.

Acknowledgments. We thank Robert Frankenstein, Björn Freiberg, Viet Nguyen, Thomas Stangneth and Katja Zatinschikow for helping with design and implementation of the user study. This work has been supported by the ESF and the Free State of Saxony.

References

1. Atterer, R., Wnuk, M., Schmidt, A.: Knowing the Users Every Move – User Activity Tracking for Website Usability Evaluation and Implicit Interaction. In: Proc. WWW (2006)
2. Carta, T., Paternò, F., de Santana, V.F.: Web Usability Probe: A Tool for Supporting Remote Usability Evaluation of Web Sites. In: Campos, P., Graham, N., Jorge, J., Nunes, N., Palanque, P., Winckler, M. (eds.) INTERACT 2011, Part IV. LNCS, vol. 6949, pp. 349–357. Springer, Heidelberg (2011)
3. Correani, F., Leporini, B., Patern, F.: Automatic Inspection-based Support for Obtaining Usable Web Sites for Vision-Impaired Users. UAIS 5(1) (2006)
4. de Vasconcelos, L.G., Baldochi Jr., L.A.: Towards an Automatic Evaluation of Web Applications. In: Proc. SAC (2012)
5. Gaedke, M., Gräf, G.: Development and Evolution of Web-Applications using the WebComposition Process Model. In: WWW9-WebE Workshop, Amsterdam (2000)
6. Gay, G.R., Li, C.Q.: AChecker: Open, Interactive, Customizable, Web Accessibility Checking. In: Proc. W4A (2010)
7. Guo, Q., Agichtein, E.: Beyond Dwell Time: Estimating Document Relevance from Cursor Movements and other Post-click Searcher Behavior. In: Proc. WWW (2012)
8. Gutschmidt, A.: Classification of User Tasks by the User Behavior. PhD thesis, University of Rostock (2012)
9. Hall, M.A.: Correlation-based Feature Subset Selection for Machine Learning. PhD thesis, University of Waikato (1998)
10. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update. SIGKDD Explor. Newsl. 11(1) (2009)
11. Hassenzahl, M.: Hedonic, emotional and experiential perspectives on product quality. In: Ghaoui, C. (ed.) Encyclopedia of Human Computer Interaction, pp. 266–272. IGI Global (2006)
12. Navalpakkam, V., Churchill, E.F.: Mouse Tracking: Measuring and Predicting Users' Experience of Web-based Content. In: Proc. CHI (2012)
13. Nebeling, M., Speicher, M., Norrie, M.C.: CrowdStudy: General Toolkit for Crowdsourced Evaluation of Web Interfaces. In: Proc. EICS (2013)
14. Nebeling, M., Matulic, F., Norrie, M.C.: Metrics for the Evaluation of News Site Content Layout in Large-Screen Contexts. In: Proc. CHI (2011)
15. Nebeling, M., Speicher, M., Norrie, M.C.: W3Touch: Metrics-based Web Page Adaptation for Touch. In: Proc. CHI (2013)
16. Nielsen, J., Molich, R.: Heuristic Evaluation of User Interfaces. In: Proc. CHI (1990)
17. Nielsen, J.: Putting A/B Testing in Its Place, `http://www.nngroup.com/articles/putting-ab-testing-in-its-place/`
18. Sauro, J.: Does Better Usability Increase Customer Loyalty? `http://www.measuringusability.com/usability-loyalty.php`
19. Speicher, M., Both, A., Gaedke, M.: TellMyRelevance! Predicting the Relevance of Web Search Results from Cursor Interactions. In: Proc. CIKM (2013)
20. Speicher, M., Both, A., Gaedke, M.: Towards Metric-based Usability Evaluation of Online Web Interfaces. In: Mensch & Computer Workshopband (2013)
21. Speicher, M., Both, A., Gaedke, M.: Was that Webpage Pleasant to Use? Predicting Usability Quantitatively from Interactions. In: Sheng, Q.Z., Kjeldskov, J. (eds.) ICWE Workshops 2013. LNCS, vol. 8295, pp. 335–339. Springer, Heidelberg (2013)
22. WaPPU Online Appendix, `http://vsr.informatik.tu-chemnitz.de/demo/WaPPU`

Evaluating Mobileapp Usability: A Holistic Quality Approach

Luis Olsina[1], Lucas Santos[1], and Philip Lew[2]

[1] GIDIS, Web Engineering School at Universidad Nacional de La Pampa, Argentina
[2] School of Computer Science and Engineering, Beihang University, China
olsinal@ing.unlpam.edu.ar, santos.ls@live.com,
philiplew@gmail.com

Abstract. As newer-generation smartphones enhance functionalities, interactions and services become more complex, leading to usability issues that are increasingly critical and challenging. Also mobile apps have several particular features that pose challenges evaluating their usability using current quality models, usability views, and their relations with target and context entities. With respect to the current literature, usability, actual usability, and user experience are poorly related to target entities (e.g. system and system in use) and context entities, to quality views (e.g. external quality and quality in use), in addition to measurement and evaluation building blocks. In this paper, we propose a holistic quality approach for evaluating usability and user experience of mobile apps. Practical use of our strategy is demonstrated through evaluation for the Facebook mobile app from the system usability viewpoint. Ultimately, a usability evaluation strategy should help designers to understand usability problems effectively and produce better design solutions so we analyze in the context of the framework's applicability toward this goal.

Keywords: Mobile app, Quality model, Usability, User Experience, Evaluation.

1 Introduction

Nowadays, for mobile apps, more robust network infrastructures and smarter mobile devices have led to increased functionality, integration and interactivity thereby warranting special attention in understanding their differences from apps on other platforms from the usability and user experience (UX) point of view because user requirements, expectations, and behavior can be somewhat different with the mobile platform. For instance, the quality design of Operability from a system viewpoint has a much different and greater influence for mobileapp Usability and UX due to the size of the screen and context of the user. Attributes such as button size, placement, color visibility, and widget usage have, for example, a much greater impact on task completion rates and task error rates [1, 5, 16] than for desktop platforms.

Nielsen *et al.* [16] indicate in recent mobile phone studies that usability varies by device category, which is mainly differentiated by screen size such as regular cellphones with small screen; smartphones with midsize screen and full A-Z keypad;

and full-screen smartphones with a nearly device-sized touch screen. Authors state that regular cellphones "*offer horrible usability, enabling only minimal interaction with websites*" (i.e. mobile webapps); and conclude "*unsurprisingly, the bigger the screen, the better the user experience when accessing websites*". This is supported by authors across several user testing studies from 2009 to 2012, in which the average success rate metric (which measures the percentage of users who were able to accomplish the proposed mobileapp tasks) rated for each mobile device category 44%, 55% and 74% respectively.

Despite these findings, the reader can ask him/herself what do "horrible usability" and "better UX" mean? What is the relationship between Usability and UX? Are they synonym concepts? Evaluating the success rate of users completing tasks correctly (as a performance indicator of effectiveness) is directly related to UX? If users are highly effective in completing tasks but they are unsatisfied due to perceived low app usefulness, then does UX score still high? Does UX depend on app Usability only or also from other characteristics such as Functional and Information Quality, Security, Reliability, and Efficiency? Is UX a quality characteristic of the system (e.g. a mobile app) or of an app in use? And, what about Usability?

Looking for the answers to these questions, we examined the current literature and found that Usability, Actual Usability (in-use) and UX are poorly linked to target entities (e.g. system and system in use) and context entities (e.g. device, environment, user, etc.), in addition to quality views (e.g. external quality and quality in use) and their quality models. Regarding quality models, ISO 25010 [9] outlines a flexible model with product/system quality –also known as internal and external quality (EQ)- and system-in-use quality –also referred to as quality in use (QinU). System quality consists of those characteristics and attributes that can be evaluated with the app in execution state both in testing and in operative stages; while system-in-use quality consists of characteristics and attributes as evaluated by end users when actually executing app tasks in a real context of use. ISO 25010 also delineates a relationship between the two quality views whereby system quality 'influences' system-in-use quality and system-in-use quality is determined by ('depends on') system quality. Usability is a system quality characteristic, while Effectiveness, Efficiency and Satisfaction are QinU characteristics. However, Actual Usability and UX, as experienced by the end user are missing concepts in the quoted standard.

From the QinU viewpoint, Hassenzahl [7] characterizes a user's goals into pragmatic, do goals and hedonic, be goals and categorizes system-in-use quality to be perceived in two dimensions, pragmatic and hedonic. Pragmatic quality refers to the system's perceived ability to support the achievement of tasks and focuses on the system's actual usability in completing tasks that are the 'do-goals' of the user. Hedonic quality refers to the system's perceived ability to support the user's achievement of 'be-goals', such as being happy, or satisfied with a focus on self.

Based on ISO 25010 among other works, such as [2, 7], we have developed 2Q2U (*Quality*, *Quality in use*, *actual Usability* and *User experience*) v2.0 [17], which ties together all of these quality concepts by relating system quality characteristics with Actual Usability and UX. Using the 2Q2U quality framework and a tailored strategy, evaluators can instantiate the quality characteristics to evaluate and conduct a systematic evaluation using the ‘depends’ and ‘influences’ relationships [14]. Besides in [12], we have addressed relevant features of mobile apps with regard to Usability and UX in the light of 2Q2U v2.0 quality models, but a global scheme which links main relationships among mobile target and context entities, quality views, characteristics and measurable properties were left for future endeavors.

Therefore, the major contributions of this research are: i) Represent relevant Usability and UX features of mobile apps with regard to system, system-in-use and context entities; ii) Analyze Usability and UX relationships, as well characteristics and attributes for mobile apps in the light of a conceptual framework and evaluation strategy; and iii) Illustrate an evaluation study for Facebook mobile app from the system usability viewpoint, showing the potential positive impact in designing quality interfaces. Lastly, we hope most of the above raised issues will be answered after reading this work.

Following this introduction, Section 2 describes a global scheme which links main relationships among mobile target and context entities, quality views, characteristics and measurable properties, measurement, and evaluation building blocks, with a focus on Usability and UX. Section 3, outlines our conceptual framework and evaluation approach which give support to the above building blocks. Section 4 demonstrates the practical use of our quality framework and evaluation strategy through the Facebook's mobileapp usability case study. Section 5 describes related work and, finally, Section 6 draws our main conclusions and outlines future work.

2 Featuring Mobileapp Usability and UX

For mobile phones, Usability and UX become crucial because users interact with apps –both native mobile apps and mobile webapps- in different contexts using devices with reduced display real estate. In particular the user’s activity at the time of usage, location, and daytime, amongst other influencing factors such as user profile and network performance have an actual impact on the quality of the user’s experience.

This section examines several features relevant for understanding and evaluating Usability and UX for mobile apps. To do this, Fig. 1 depicts the main building blocks which link some relationships among: i) entity categories, quality views/characteristics, and measurable properties (green/orange boxes); ii) measurement (light-blue box); and iii) evaluation (pink box). Next, we examine particularly i) features which allow better understanding non-functional requirements to further specify Usability and UX attributes for interface-, task-, and perception-based evaluations. First, we give a summary followed by deeper discussion.

The 'entity' label in the upper box represents the potential target entity category to be evaluated. It is defined as the "*object category that is to be characterized by measuring its attributes*", while an attribute is "*a measurable physical or abstract property of an entity category*" [18]. There are two instances of (super) categories for target entities that are of interest for evaluating Usability and UX, viz. product/system and system in use. In turn, an entity category can aggregate sub-entities, e.g., a mobile app is composed of basic and advanced GUI objects from the interface standpoint, as shown in Fig. 1. Moreover, lower level entities can be identified for GUI objects like button, menu, widget, etc. Another label in the upper box is 'context' which is defined as "*a special kind of entity representing the state of the situation of a target entity category, which is relevant for a particular information need*". So, system in use is characterized by a context-in-use entity (upper-right orange box) which in turn can aggregate environment, user and task contextual sub-entities.

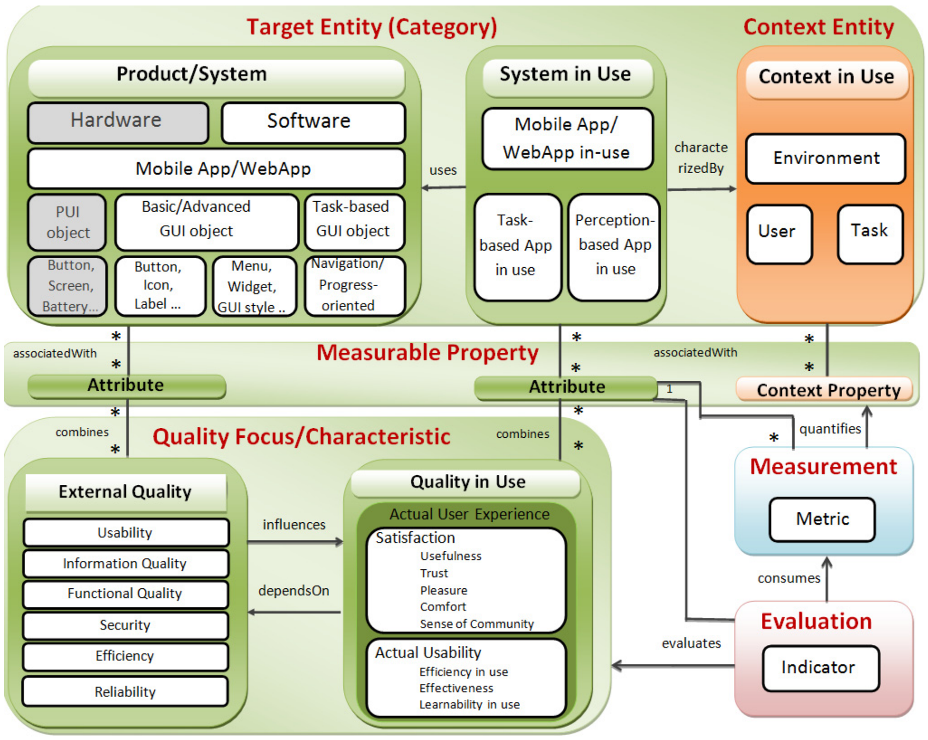


Fig. 1. Global scheme which links the main relationships among mobile Target and Context Entities, Quality Focuses, Measurable Properties, Measurement, and Evaluation building blocks. Note that the two lower-level Product/System sub-entity categories are just addressed for the Usability characteristic, which deals with user interface-oriented evaluation issues –PUI/ GUI stands for Physical/Graphical User Interface.

On the other hand, an entity (category) and their sub-entities cannot be measured and evaluated directly but only by means of the associated measurable properties, i.e. attributes and context properties accordingly (see Fig. 1). Quality models can be the focus for different entities, and usually specify product/system or system-in-use quality requirements regarding main characteristics that can be further subdivided into sub-characteristics, which combine attributes. Product/system quality requirements are modeled by the EQ focus (view), which includes higher-level characteristics such as Usability, Security, Functional Quality, etc. Instead, system-in-use quality requirements are modeled by the QinU view, which include higher-level characteristics such as Actual UX, Satisfaction and Actual Usability.

Lastly, looking at the entity building block relationships, we see the ‘uses’ and ‘characterized by’ relations. Also between EQ and QinU views, we observe that system quality ‘influences’ system-in-use quality or system-in-use quality ‘depends on’ system quality. Note that for instantiated EQ and QinU models these relationships can be explored in light of concrete entity attributes by performing evaluations. For instance, using an evaluation strategy we can explore relationships between system quality and system-in-use quality attributes that may contribute to usage improvements. Regarding the above global scheme, in the next three sub-sections we closely examine the features of mobileapp Usability, UX, and context.

2.1 Featuring Mobileapp Usability

Usability is a characteristic for a system from the EQ viewpoint. It is one out of eight EQ characteristics in 2Q2U v2.0 (see [17] for quality models details). We define Usability as the "*degree to which the product or system has attributes that enable it to be understood, learned, operated, error protected, attractive and accessible to the user, when used under specified conditions*". (Note this definition is very close to that in ISO 9126-1 [11] rather than to [9], as we discuss in related work).

Examining the first part of the above definition, products are entities at early phases of a software life cycle (e.g., textual or graphical documents, etc.); while systems are executable software products (e.g. a mobile app in a testing or operative stage), which could include hardware and software together. Examining the second part of the above definition, we observe that the system (particularly, interface-related objects of the app) has attributes that enable the user to interact considering certain factors. These are the Usability sub-characteristics which can be evaluated through Understandability, Learnability, Operability, User error protection, UI aesthetics and Accessibility. Table 1 shows the Usability sub-characteristics and attributes definitions used in the Facebook evaluation study, in Section 4.

Recalling that characteristics and sub-characteristics combine attributes which are associated to entities (see Fig. 1) some typical mobileapp sub-entities that should be considered for Usability design and evaluation are entry fields and widgets, menus, carousels, breadcrumb path, amongst others. Entity sub-categories specific for Usability evaluation can be physical and graphical user interface (PUI/GUI) objects [6], and task-based GUI objects. A possible categorization for GUI objects can be basic or advanced objects (similar to that described in [8]).

All definitions for sub-characteristics and attributes in Table 1 include to a great extent the referred target sub-entity. For instance, the Visibility (1.3.2 coded) sub-characteristic is defined as "*degree to which the application enables ease of operation through controls and text which can be seen and discerned by the user in order to take appropriate actions*", and one combined attribute viz. Brightness difference appropriateness (1.3.2.1.1) is defined as "*degree to which the foreground color of the GUI object (e.g. text, control, etc.) compared to the background color provide appropriate brightness difference*". Actually, many attributes can determine whether or not the application is easily visible to the user. Depending on the context, different text colors and backgrounds can have a positive or negative impact. Remember that mobile users want to glance quickly and understand and operate almost immediately and there may be glare on their screen if they are outdoors. Also this means that appropriate usage of control/text colors (and size) can greatly impact the user's speed of comprehension and therefore, operational effectiveness and efficiency.

In addition to Usability, characteristics to evaluate other mobileapp EQ aspects are Security, Functional and Information Quality, etc. in which the target sub-entities should be defined accordingly, at least to the system lower layers.

Table 1. Definition of EQ/Usability sub-characteristics and attributes –in *italic*

Characteristic/*Attribute*	**2Q2U v2.0 Definition**
1 Usability	Degree to which the product or system has attributes that enable it to be understood, learned, operated, error protected, attractive and accessible to the user, when used under specified conditions.
1.1 Understandability (synonym Appropriateness Recognizability)	Degree to which users can recognize whether a product or system is appropriate for their needs. Note: Same ISO 25010 definition.
1.1.1 Familiarity	Degree to which the user understand what the application, system's functions or tasks are about, and their functionality almost instantly, mainly from initial impressions
1.1.1.1 Global organization scheme understandability	Degree to which the application scheme or layout is consistent and adheres to either de facto or industry standard to enable users to instantly understand its function and content.
1.1.1.2 Control icon ease to be recognized	Degree to which the representation of the control icon follows or adheres to an international standard or agreed convention.
1.1.1.2.1 Main control icon ease to be recognized	Degree to which the representation of the main controls icons follows or adheres to an international standard or agreed convention.
1.1.1.2.2 Contextual control icon ease to be recognized	Degree to which the representation of the contextual controls icons follows or adheres to an international standard or agreed convention.
1.1.1.3 Foreign language support	Degree to which the application functions, controls and content has multi-language support enabling user to change his/her language of preference.
1.2 Learnability	Degree to which the product or system enables users to learn its app.
1.2.1 Feedback Suitability	Degree to which mechanisms and information regarding the success, failure or awareness of actions is provided to users to help them interact with the application.
1.2.1.1 Current location feedback appropriateness	Degree to which users are made aware of where they are at the current location by an appropriate mechanism.
1.2.1.2 Alert notification feedback appropriateness	Degree to which users are made aware of new triggered alerts that they are involved by an appropriate mechanism.
1.2.1.3 Error message appropriateness	Degree to which meaningful error messages are provided upon invalid operation so that users know what they did wrong, what information was missing, or what other options are available.
1.2.2 Helpfulness	Degree to which the software product provides help that is easy to find, comprehensive and effective when users need assistance
1.2.2.1 Context-sensitive help appropriateness	Degree to which the application provides context sensitive help depending on the user profile and goal, and current interaction.
1.2.2.2 First-time visitor help appropriateness	Degree to which the application provides an appropriate mechanism (e.g. a guided tour, etc) to help beginner users to understand the main tasks that they can do.
1.3 Operability	Degree to which a product or system has attributes that make it easy to operate and control. Note: Same ISO 25010 definition
1.3.1 Data Entry Ease	Degree to which mechanisms are provided which make entering data as easy and as accurate as possible.

Table 1. (*Continued*)

1.3.1.1 Defaults	Degree to which the application provides support for default data.
1.3.1.2 Mandatory entry	Degree to which the application provides support for mandatory data entry.
1.3.1.3 Widget entry appropriateness	Degree to which the application provides the appropriate type of entry mechanism in order to reduce the effort required.
1.3.2 Visibility (synonym Optical Legibility)	Degree to which the application enables ease of operation through controls and text that can be seen and discerned by the user in order to take appropriate actions.
1.3.2.1 Color visibility appropriateness	Degree to which the main GUI object (e.g. text, control, etc.) color compared to the background color provide sufficient contrast and ultimately appropriate visibility.
1.3.2.1.1 Brightness difference appropriateness	Degree to which the foreground color of the GUI object (e.g. text, control, etc.) compared to the background color provide appropriate brightness difference.
1.3.2.1.2 Color difference appropriateness	Degree to which the foreground text or control color compared to the background color provide appropriate color difference.
1.3.2.2 GUI object size appropriateness	Degree to which the size of GUI objects (e.g. text, buttons, and controls in general) are appropriate in order to enable users to easily identify and operate them.
1.3.2.2.1 Control (widget) size appropriateness	Degree to which the size of GUI controls are appropriate in order to enable users to easily identify and operate them.
1.3.2.2.2 Text size appropriateness	Degree to which text sizes and font types are appropriate to enable users to easily determine and understand their meaning.
1.3.3 Consistency	Degree to which users can operate the task controls and actions in a consistent and coherent way even in different contexts and platforms.
1.3.3.1 Permanence of controls	Degree to which main and contextual controls are consistently available for users in all appropriate screens or pages.
1.3.3.1.1 Permanence of main controls	Degree to which main controls are consistently available for users in all appropriate screens or pages.
1.3.3.1.2 Permanence of contextual controls	Degree to which contextual controls are consistently available for users in all appropriate screens or pages.
1.3.3.2 Stability of controls	Degree to which main controls are in the same location (placement) and order in all appropriate screens.
1.4 User Error Protection	Degree to which a product or system protects and prevents users against making errors and provides support to error tolerance.
1.4.1 Error Management	Degree to which users can avoid and recover from errors easily.
1.4.1.1 Error prevention	Degree to which mechanisms are provided to prevent mistakes.
1.4.1.2 Error recovery	Degree to which the application provides support for error recovery.
1.5 UI Aesthetics (synonym Attractiveness)	Degree to which the UI enables pleasing and satisfying interaction for the user. Note: Same ISO 25010 definition.
1.5.1 UI Style Uniformity	Degree to which the UI provides consistency in style and meaning.
1.5.1.1 Text color style uniformity	Degree to which text colors are used consistently throughout the UI with the same meaning and purpose.
1.5.1.2 Aesthetic harmony	Degree to which the UI shows and maintains an aesthetic harmony regarding the usage and combination of colors, texts, images, controls and layouts throughout the whole application.

Table 2. Definition of QinU (sub-)characteristics absent in [9] or were rephrased in 2Q2U v2.0

Characteristic/Sub-characteristic Definition	**ISO 25010 QinU Definition**
Actual User Experience: Degree to which a system in use enable specified users to meet their needs to achieve specific goals with satisfaction, actual usability, and freedom from risk in specified contexts of use	Note: Absent characteristic in ISO 25010, but similar definition to QinU in this standard
Actual Usability (synonym Usability in use)*:*Degree to which specified users can achieve specified goals with effectiveness, efficiency, learnability in use, and without communicability breakdowns in specified contexts of use	Note: Absent characteristic, but similar concept (i.e. *usability in use*) was in the ISO 25010 draft, and in [2]
Effectiveness: Degree to which specified users can achieve specified goals with accuracy and completeness in specified contexts of use	*Effectiveness*: Accuracy and completeness with which users achieve specified goals
Efficiency (**in use**): Degree to which specified users expend appropriate amounts of resources in relation to the effectiveness achieved in specified contexts of use	*Efficiency:* Resources expended in relation to the accuracy and completeness with which users achieve goals
Learnability (**in use**): Degree to which specified users can learn efficiently and effectively while achieving specified goals in specified contexts of use	Note: Absent characteristic
Sense of Community: Degree to which a user is satisfied when meeting, collaborating and communicating with other users with similar interest and needs	Note: Absent characteristic

As we depict in the next sub-section, UX is a broader concept that depends not only on Usability but also on other system characteristics such as Functional and Information Quality, Security, Reliability, Efficiency, and contexts of use as well.

2.2 Featuring Mobileapp UX

Fig. 1 shows UX as the higher-level characteristic for QinU evaluations. The QinU view characterizes the impact that the system in use (e.g. a mobile app) has on actual users in real contexts of use, i.e., while users perform application tasks in a real environment. Actual UX is defined in Table 2, as "*degree to which a system in use enable specified users to meet their needs to achieve specific goals with satisfaction, actual usability, and freedom from risk in specified contexts of use*".

UX is determined by the satisfaction of the user's be goals (hedonic), and do goals (pragmatic) as noted by Hassenzahl [7]. Moreover, do-goals relate to the user being able to accomplish what they want with Effectiveness and Efficiency (i.e. Actual Usability or Usability in use), while be-goals relate to the user's satisfaction. Satisfaction in [9] includes those subjective, perception-oriented sub-characteristics including Usefulness, Trust, Pleasure, and Comfort -also Sense of Community in [17].

Ultimately, Usability deals with the specification and evaluation of interface-based sub-characteristics and attributes of a system, while Actual Usability deals with the specification and evaluation of task-based sub-characteristics and attributes of an app in use, and Satisfaction with perception-based sub-characteristics and attributes. Recalling that sub-characteristics combine attributes which are associated to entities, we have considered in Fig. 1 two typical app-in-use target sub-entities, namely: Task-based and Perception-based App in-use. The Task-based App-in-use sub-entity can be evaluated using Effectiveness, Efficiency and Learnability in-use attributes. In [14], for the JIRA webapp in-use, we evaluated the "Entering a new defect" task performed by 50 beginner tester users, in which for example Effectiveness combined three attributes such as Sub-task correctness, Sub-task completeness and Task successfulness. (Note that Task successfulness attribute was measured in a similar way that the Average success rate used by Nielsen *et al.* [16]). On the other hand, the Perception-based App-in-use sub-entity can be measured and evaluated using Satisfaction sub-characteristics and attributes that can be included in questionnaire items as in [13], or evaluated through other methods such as observation.

As a final remark, mobileapp selected tasks should be evaluated with respect to real users performing real tasks. This issue includes several key design concerns that have significant impact on the Effectiveness and Efficiency of the final user. For example, *task workflows* need to be designed with the most common tasks in mind that would be suited to mobile usage. Because of the context of use of a mobile user, and the mobile user's limited attention span, the choice of tasks, task workflow and length are extremely important for this limited task set. If task workflows are not designed to be short, there is a higher probability of user error and a lower rate of completion –see Effectiveness definition in Table 2. Workflows therefore need to be compressed by combining several steps into one through careful task definition, evaluation and analysis. Reduced workflows, in turn, reduce task times and increase Efficiency (see definition in Table 2) while, at the same time, reducing error rates and error rate reduction is extremely critical for users with short attention spans. If you are driving and executing a task and get an error, do you continue trying?

2.3 Featuring Mobileapp Context

As mentioned above, the Context entity (category) is a special kind of entity representing the state of the situation of a target entity to be assessed, which is relevant for a particular measurement and evaluation (M&E) information need. Context for a given QinU M&E project is particularly important –i.e. a must-, as instantiation of QinU requirements must be done consistently and in the same context so that evaluations and improvements can be accurately assessed and compared. But also context is important regarding the EQ view, as we describe in Section 4. (Note that in order to reduce clutter in Fig 1, we did not draw an upper-left orange box for product/system context). For instance, system in use is characterized by a context-in-use entity, which in turn can aggregate environment, user and task sub-entities, while a product/system context (for idem target entity), can be characterized by sub-entities such as device (hardware), software, etc.

As commented in [12], the context of a mobileapp user is much different than a traditional desktop webapp user not only due to the size of the screen but also to other situations that influence the user's environment and therefore its behavior. In particular the user's activity at the time of usage, location, amongst other influencing factors such as user profile have actual impact on the quality of the user's experience.

Some a few of these factors considering context sub-entities and properties include: i) *Activity*: What users are doing at the time of usage have a significant influence on the user's attention span; e.g. if they are driving, then they have a very short attention span, maybe 1 second, versus if they are in the middle of a conversation, perhaps they have an attention span of 3 seconds; ii) *Day/time of day*: The day and time can impact what a user is doing, and the level of natural light. Unlike desktop apps which are typically accessed indoors, the usage of mobile apps is particularly sensitive to this contextual factor influencing visibility; ii) *Location*: The location of the user influences many elements; e.g. indoors, outdoors, in a car or in an elevator, all of which can also be related to the user activity; iii) *Network performance*: Obviously the speed at which an app uploads and downloads data is going to have a great impact because of the decreased attention span; iv) *User profile*: The increasing complexity of software combined with an aging user demographic has an interesting effect on the usability of mobile apps. For aging users, usually their close range vision capability has diminished along with their dexterity. On the other hand, apps have become complex, and therefore function and content simplicity and understandability are also critical and influenced by the particular user group. Not only are there more aging users, there are also more younger users as children these days begin using computing devices as toddlers; v) *Device*: The size and type of the device and its physical features influence what the user can see (or not see) as well as the placement and number of controls and widgets in reduced real-estate displays.

This shortage of resources and particular contexts of use all impact on the UX. Lastly, context properties are not part of the EQ or QinU models, but should be recorded accordingly for characterizing the situation of the target entities at hand.

3 Conceptual Framework and Evaluation Approach

3.1 M&E Conceptual Framework

At this point, it is worth mentioning that the main building blocks depicted in Fig. 1 are grounded in a M&E conceptual framework. We have built –as part of evaluation strategies- the C-INCAMI (*Contextual-Information Need*, *Concept model*, *Attribute*, *Metric* and *Indicator*) conceptual framework [18], which is structured in six components, namely: (a) Measurement and Evaluation Project; (b) Non-functional Requirements; (c) Context; (d) Measurement; (e) Evaluation; and (f) Analysis and Recommendation. Each component contains key terms and relationships. Fig 2 shows, for illustration purpose, the (b), (c), and (d) components whose colors match those green/orange/light-blue boxes of Fig. 1.

In fact, the different labels in Fig 1 are mostly instances of the concepts, properties, and relationships included in the C-INCAMI conceptual framework. For instance "System" and "System in Use" in Fig. 1 are two instances of the *Entity Category* term; specifically, each string is the value of the *name* field in Fig. 2. Since an entity category can have *sub-entity* categories, "Basic/Advanced GUI object", "Menu", etc. are instances of sub-entity categories. *Entity* term represents a concrete object; for example, "Facebook mobile app" is the entity *name* that *belongs to* the "System" category regarding the EQ *focus*.

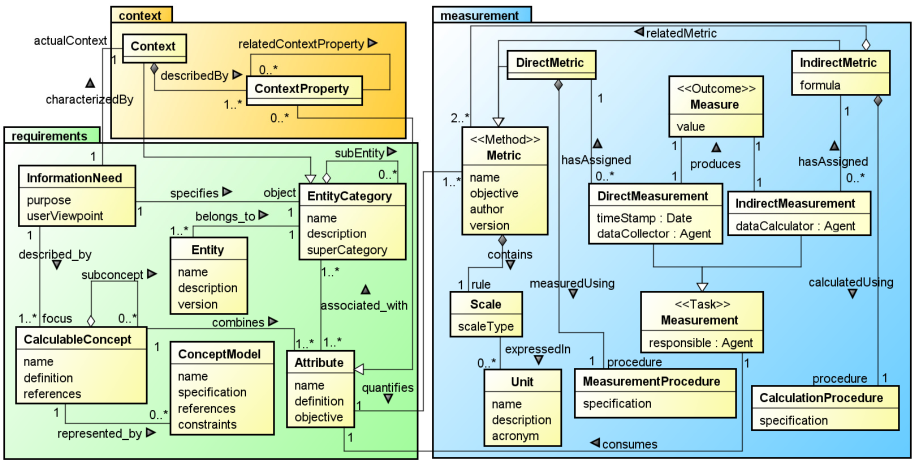


Fig. 2. C-INCAMI Nonfunctional Requirements, Context, and Measurement components

Therefore, the *requirements* component specifies the *Information Need* for a M&E project, i.e., the *purpose* (e.g. "understand", "improve") and the *user Viewpoint* (e.g. "final user", "developer"). In turn, it *focuses* on a *Calculable Concept* (i.e. characteristics whose *names* are for example "External Quality", "User Experience", etc.) and *specifies* the *Entity Category* to be evaluated. On the other hand, a calculable concept and its *sub-concepts* (e.g. "Usability") can be *represented by* a *Concept Model* (e.g. an "EQ model") where the leaves of an instantiated quality model are *Attributes* which are *associated with* a target entity. Table 1 specifies the requirements tree for "Usability", which contains the *names* and *definitions* for the selected sub-characteristics and attributes used in the Facebook mobileapp evaluation.

The *context* component (in Fig. 2) shows explicitly that *Context* is a special kind of *Entity Category*. Context represents the state of the situation of a target entity, which is relevant for a particular information need. To describe the context sub-entities (e.g. "Environment", "Device", etc.) *Context Properties* are used, which are also attributes. Additionally, attributes –as measurable properties- can be *quantified* by metrics and interpreted by indicators.

Metric is a key term in the *measurement* component in Fig 2 (see also Fig. 1). This component allows specifying *Direct* and *Indirect Metrics* used by *Direct* and *Indirect Measurement* tasks which produce *Measures.* A metric is "*the defined measurement or calculation procedure and the scale*". So a metric represents the *how*, that is to say, the method that should be assigned to the steps of a measurement task (the *what*). Lastly, a *Measure* is the number or category assigned to an attribute by making a measurement upon a concrete entity. In order to illustrate the added value of a well-defined measurement component, Table 3 shows the derived template for indirect and direct metric specifications to the "Permanence of main controls" attribute. The "Operability" sub-characteristic combines this attribute that is coded 1.3.3.1.1 in Table 1. Additionally, the screenshot in Fig. 3.b shows the concrete sub-entity named "Main controls bar" that can be further measured.

Table 3. Indirect and direct metric specifications to the *Permanence of main controls* attribute

Target Entity Category: **Name**: *System*; **Sub-Entity Category: Name**: *Smartphone mobile app*;
Concrete Entity: **Name**: *Facebook app*; **Version**: 3.8; **Sub-Entity Description:** Set of *Screens* of the *Facebook app* where the *Main controls bar* is (or should be) containing the set of *Main controls (Buttons)*

Attribute: **Name**: *Permanence of main controls*; **Code**: *1.3.3.1.1* in Table 1
Definition: Degree to which main controls are consistently available for users in all appropriate screens or pages; **Objective**: To determine the degree to which the main controls are present in all appropriate screens.

Indirect Metric: **Name**: *Ratio of Main Controls Permanence* (%MCP); **Objective**: To determine the percentage of permanence for controls from the set of main controls in the application selected screens;
Author: Santos L.; **Version**: 1.0;
Calculation Procedure: Formula: $\%\mathrm{MCP} = \left[\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}\mathrm{MCPLij}}{(m*n)}\right] * 100$; for i=1 to m and j=1 to n, where m is the number of application main controls and n is the number of application selected screens; with m, n > 0
Numerical Scale: Representation: Continuous; **Value Type**: Real; **Scale Type**: Ratio;
Unit Name: Percentage; **Acronym**: %
Related Metrics: Main control permanence level (MCPL)

Related Direct Metric: **Name**: *Main Control Permanence Level* (MCPL); **Objective:** To determine the permanence level of a selected control in a given application screen; **Author**: Santos L.; **Version**: 1.0;
Measurement Procedure: Type: Objective; **Specification**: The expert inspects the main controls bar in a given screen in order to determine whether the button is available or not, using the 0 or 1 allowed values. Where 0 means the main button is absent in the screen, and 1 means the main button is present in the screen;
Numerical Scale: Representation: Discrete; **Value Type**: Integer; **Scale Type**: Absolute;
Unit: Name Control

3.2 Evaluation Approach and Strategies

This sub-section gives a summary of our generic evaluation approach, which is made up of a *quality modeling framework* and *M&E strategies*, where a concrete strategy should be selected for purposefully instantiating quality models, processes, and performing evaluations for a concrete project information need. Particularly, the generic evaluation approach relies on two pillars, namely: i) a *quality modeling framework* –where 2Q2U v2.0 is a subset [17], which includes the EQ and QinU views and the 'depends' and 'influences' relationships between them; and ii) *M&E strategies*, which in turn are based on three principles viz. a *M&E conceptual framework* (as introduced in the previous sub-section), *process view specifications*, and *method specifications*.

So far, we have developed two integrated strategies which include these three principles, namely: GOCAME (*Goal-Oriented Context-Aware Measurement and Evaluation*) [17, 18], and SIQinU (*Strategy for understanding and Improving Quality in Use*) [14]. GOCAME is a multi-purpose strategy that follows a goal-oriented and context-based approach in defining and performing M&E projects. GOCAME is a multi-purpose strategy because it can be used to evaluate (e.g. "understand", "improve", etc.) quality not only for product, system and system-in-use entities but also for others such as resource, by using their instantiated quality models and tailored process accordingly. However, GOCAME does not incorporate the QinU/EQ/QinU relationships and improvement cycles as in SIQinU. Rather it can be used to understand the current or future situation, as an evaluation snapshot, of concrete entities. On the other hand, SIQinU is a specific-purpose strategy, which has specific processes, methods and change procedures that are not specified in GOCAME. Ultimately, given the target information need and objective, we can select the specific strategy and its tailored processes and methods in order to fulfill that specific goal.

For example, GOCAME has a well-defined M&E process specification, which is composed of six generic activities, namely: (A1) *Define Non-functional Requirements*; (A2) *Design the Measurement*; (A3) *Implement the Measurement*; (A4) *Design the Evaluation*; (A5) *Implement the Evaluation*; and (A6) *Analyze and Recommend*. Each activity can be accordingly tailored for a specific quality focus regarding the information need, e.g. if the focus is on EQ then A1 is named *Define Non-functional Requirements for EQ*, and so on. Instead, if the focus is on QinU then A1 is named *Define Non-functional Requirements for QinU*, and so forth. Note that in our process specifications each activity is not atomic, so it should be decomposed into tasks.

Lastly, the strategies' activities are supported by different method specifications. Since the M&E strategies rely on the quality modeling framework which is made up of quality models, inspection of characteristics and attributes is the basic method category. Attributes are supported by metric and elementary indicator method specifications, while quality models are calculated using different indicator aggregation methods such as LSP (*Logic Scoring of Preference*) [4], which is a weighted multi-criteria aggregation method. However, user testing and inquiry method categories can be used –mainly for QinU- meanwhile attributes of Efficiency, Effectiveness, Learnability in use and Satisfaction can be derived from task usage log files, questionnaire items, etc., as we did in [13, 14]. For planning and performing changes traditional methods and techniques such as refactoring, re-structuring, re-parameterization, among others can be used as well. The next section demonstrates a practical use of our quality framework and GOCAME strategy through excerpts of our Facebook's mobileapp Usability evaluation study.

4 Usability Evaluation for the Facebook Mobile App

The abovementioned A1 activity named *Define Non-functional Requirements for EQ* has a specific goal or problem as input and a nonfunctional specification document as output. A1 consists of: *Establish EQ Information Need* (A1.1), *Select an EQ Model* (A1.2), and *Specify (System) Context* (A1.3) sub-activities [17].

Considering A1.1, the *purpose* of the information need is to "understand" the current EQ satisfaction level achieved, particularly by evaluating the "Usability" strengths and weaknesses from the "final user" *viewpoint.* "Facebook mobile app" is the concrete entity whose sub-entities for the Usability *focus* are related to basic-, advanced- and task-based GUI objects (recall Fig. 1). For example, in the Fig. 3.b screenshot the "Main controls bar" contains a set of main controls or "Buttons".

For the given focus, the A1.2 sub-activity allows selecting from a repository the sub-characteristics and attributes to be included. Table 1 documents the resulting requirements tree, which includes "Understandability" (1.1), "Learnability" (1.2), "Operability" (1.3), "User error protection" (1.4), and "Aesthetics" (1.5) sub-characteristics. For example, "Operability" includes in turn sub-characteristics such as "Visibility" (1.3.2), "Consistency" (1.3.3), etc., which *combine* attributes *associated* to the entities. Particularly, "Color visibility appropriateness" (1.3.2.1) combines two attributes associated to "Color/Text" objects (see Fig.3.a); while "Permanence of controls" (1.3.3.1) combines two attributes associated to "Main controls bar" and "Contextual control" objects (Fig.3.a and b). For instance, the "Permanence of main controls" attribute (1.3.3.1.1) is defined in Table 1 as "*degree to which main controls are consistently available for users in all appropriate screens or pages*". Note that in the main controls bar of the three shown screenshots all buttons are not always available, so the measured value will be produced in the A3 (*Implement the EQ Measurement*) activity, using the appropriate metric from Table 3.

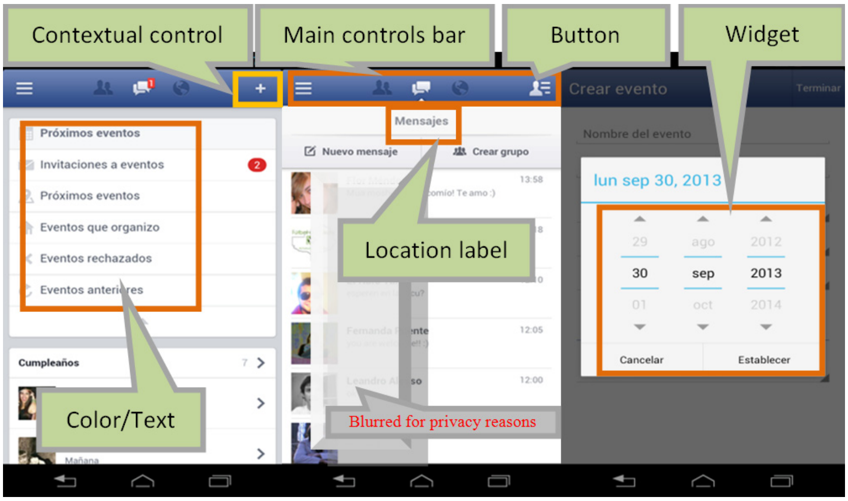


Fig. 3. Three Facebook screenshots: a) Contextual control and Color/Text objects are highlighted; b) The Main controls bar, and the chat Button, which is not available in the other two screenshots; and c) A typical date widget used when create event task is performed

Lastly, the A1.3 sub-activity deals with the selection of *context properties* (and further values) like "mobile device type" (e.g. tablet, mobile phone); "mobilephone generation" (e.g. regular cellphone, mid-sized smartphone, full-sized smartphone [16]); "mobilephone device brand-model"; "target mobileapp type" (e.g. native mobile app, mobile webapp), among many others, recalling that *context* is a special kind of entity related to the entity category to be evaluated, as mentioned in sub-section 2.3.

Once the information need, EQ requirements, and context specifications were yielded, the next A2 activity, *Design the EQ Measurement*, consists of selecting the meaningful metrics from a repository to quantify the 23 measurable attributes. One direct or indirect metric should be assigned per each attribute of the requirements tree respectively. For example, the "Ratio of Main Controls Permanence" (%MCP) *indirect metric* whose *objective* is "*to determine the percentage of permanence for controls from the set of main controls in the application selected screens*" was chosen to quantify the 1.3.3.1.1 attribute, as shown in Table 3. While an indirect metric has a *calculation procedure* for its *formula* specification, a direct metric has a *measurement procedure*. %MCP includes a related direct metric where the measurement procedure indicates "*the expert inspects the main controls bar in a given screen in order to determine whether the button is available or not, using the 0 or 1 allowed values. Where 0 means the main button is absent in the screen, and 1 means the main button is present in the screen*". In summary, 48 metrics were designed for this study taking into account direct, indirect and related metrics for the latter.

The A3 activity produces the *measures* for all metrics at given moments in time as well the linked data to concrete object references and parameters. Data collection for metrics on the Facebook app (ver.3.8 for Android) were performed from Dec 26-28, 2013. The measure for %MCP gave 54.9% of permanence of main controls, regarding that 5 main buttons should be placed in the main controls bar in 35 appropriate screens (out of 38 app screens). Looking at Fig 3, we can observe for example that the "chat" button is absent in screens a) and c), so if the end user wants to trigger this action on those screens, he/she needs to perform more clicks than needed to initiate the task.

Table 4. Excerpt of *Usability* sub-characteristics and attributes from Table 1. Only *Operability* sub-characteristics and attributes are fully shown with Elementary Indicator values (2nd column); the 3rd column shows Partial/Global Indicator values, which are all in % scale unit.

1 Usability		**60.5**
1.1 Understandability		**76.1**
1.2 Learnability		**59.7**
1.3 Operability		**80.7**
1.3.1 Data Entry Ease		90
1.3.1.1 Defaults	100	
1.3.1.2 Mandatory entry	50	
1.3.1.3 Widget appropriateness	100	
1.3.2 Visibility (synonym **Optical Legibility**)		81.5
1.3.2.1 Color visibility appropriateness		100
1.3.2.1.1 Brightness difference appropriateness	100	
1.3.2.1.2 Color difference appropriateness	100	
1.3.2.2 GUI object size appropriateness		63
1.3.2.2.1 Control (widget) size appropriateness	100	
1.3.2.2.2 Text size appropriateness	42.1	
1.3.3 Consistency		75.5
1.3.3.1 Permanence of controls		57.3
1.3.3.1.1 Permanence of main controls	54.9	
1.3.3.1.2 Permanence of contextual controls	67.4	
1.3.3.2 Stability of controls	95.5	
1.4 User Error Protection		**8.4**
1.5 UI Aesthetics		**80.8**

Once metrics were selected for quantifying all attributes, then A4 can be performed, which deals with designing the EQ evaluation. For space reasons, we did not describe the *evaluation* component in sub-section 3.1, but a key concept is *Indicator*, as shown in Fig. 1. While an *elementary indicator* evaluates the satisfaction level met for an elementary requirement, i.e., an attribute of the requirements tree, a *partial/global indicator* evaluates the satisfaction level achieved for partial (sub-characteristic) and global (characteristic) requirements represented in the quality model. Therefore, a new *scale* transformation and *decision criteria* (in terms of *acceptability levels* and ranges) are defined. In this study, we used three acceptability ranges in a percentage scale: a value within 60-80 (a marginal –yellow- range) indicates a need for improvement actions; a value within 0-60 (an unsatisfactory –red- range) means change actions must take place with high priority; and a score within 80-100 indicates a satisfactory level –green- for the analyzed attribute or characteristic.

Details of elementary and global evaluation, as well as the LSP model used in this study to calculate indicators (A5 activity) can be referred elsewhere [18]. Table 4 shows the elementary and partial indicators' values for "Operability", and only partial and global indicators' values for the other sub-characteristics in addition to the acceptability levels achieved.

Finally, GOCAME projects record all data, metadata and information coming from metrics and indicators as well as the quality model and context specifications and values. The *Analyze and Recommend* (A6) activity produces a recommendation document, which can facilitate planning actions for further improvement.

Based on indicator results shown in the 3rd column of Table 4, we can observe that the Usability characteristic in the Facebook app reached a marginal acceptability level (60.5%), which means a need for improvement actions. Taking into account its sub-characteristics viz. Understandability (1.1), Learnability (1.2) and User Error Protection (1.4) reached a marginal and unsatisfactory acceptability levels respectively. Therefore some of their elementary indicators are performing weakly and surely need recommendations for attribute changes.

On the other hand, Operability (1.3) met the satisfactory level of 80.7%. However, this does not imply that there are no weakly performing attributes for Operability. While Color Visibility Appropriateness (1.3.2.1) scored 100 in its two attributes, Permanence of Controls (1.3.3.1) scored in its two attributes, 54.9 and 67.4 respectively (see 2nd column), so a recommendation for further improvement can be made. For understanding the reasons and planning change actions, the metric specification and the measured values are central in GOCAME for these endeavors. %MCP metric allowed (in A3) to store per each main button its availability in each corresponding app screen in which must stay. So evaluators can easily understand, for instance, where the "chat" button is absent, and so for each button of the main controls bar. Also, the tailored strategy may help designers to understand and act on (system) usability problems effectively to produce better design solutions as well.

Finally, as the reader can surmise the metric design specification helps not only planning the change action, but also gauging (predicting) the improvement gain once the action is performed. Ultimately, if we add a new activity to the six described GOCAME activities (as we did it in fact previously with SIQinU), e.g. (A7) *Plan and Perform Improvement Actions*, then the A3, A5 and A6 can be fully reused for re-evaluation and analysis of the improvement gain with regard to the previous app version. Note that changes should be made on the app entity not on the app in use.

5 Related Work and Discussion

As commented previously, in the state-of-the-art literature, Usability, Actual Usability and UX features are very often poorly linked to target entities (e.g. system and system in use) and context entities (e.g. device, environment, user, etc.), in addition to EQ and QinU views and their relationships. Bevan [3] states that international standards for Usability should be more widely used because one of their main purposes is to impose consistency, compatibility, and safety. Usability has also been integrated into standards for software quality and evaluation; e.g. ISO 25010 (which supersedes to ISO 9126-1 [11]) provides a comprehensive structure for the role of Usability as part of system quality as well a broader concept of QinU increasing the business relevance of usability in many situations. Besides, author indicates that referring to the terminology from the field of software quality, it can be said that UX is more related to the concept of QinU, whereas Usability more to EQ.

From our viewpoint, one of the strengths in ISO 25010 is not only the quality models and included characteristics but also the two quality views and relationship whereby system quality ‘influences’ system-in-use quality and system-in-use quality ‘depends on’ system quality. However, some weaknesses we point out are: UX is still an absent characteristic in ISO; there exists a dual Usability definition which blurs system Usability with QinU meanings (i.e. Usability is defined in [9] as "*degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use*", while QinU as "*degree to which a product or system can be used by specific users to meet their needs to achieve specific goals with effectiveness, efficiency, freedom from risk and satisfaction in specific contexts of use*") so, we consider the Usability definition given in [11] (adapted in Table 1) is closer to the intended aim; and, the Context coverage characteristic included in the [9] QinU model, which can be represented independently of quality models, as shown in previous sections and in [17]. Therefore, in order to bridge this gap, we have developed the 2Q2U v2.0 quality modeling framework, considering also contributions such as [2, 7], amongst others.

On the other hand, in the Apple [1] and Google [5] design and user interface guidelines, the relationship between mobileapp entities with Usability and UX concepts is not definitively explicit in models, nor is it represented in the Usability works in [16], nor in other quality-related research such as [15, 19]. For instance, Nielsen *et al.* list out in [16] many features and checklists of mobile apps in that would be desired or needed in certain contexts of use but do not use quality views and modeling approaches. Therefore, the capability for consistent application using a conceptual framework and strategies to systematically apply concepts and evaluate and improve a mobile app is rather limited. (Recall the raised issues in Section 1).

Lastly, a holistic approach similar to ours for evaluating the Usability of mobilephones in an analytical way is documented in [8], which is based on a multi-level, hierarchical model of Usability factors. These factors related to views and entities are collectively measured to give a single score with the use of checklists. Moreover, the conceptual framework and strategy involves a hierarchical model of usability factors, four sets of checklists, a quantification method, and an evaluation process. The conceptual framework for usability indicators [6] is based on the ISO 15939 [10] measurement model, which its terms are structured in a glossary. Conversely, we developed an ontology [18] for the C-INCAMI M&E components (recall Fig. 2) where [10] was one of the used sources. Consequently, from components we derive metric an indicator metadata in templates (as in Table 3) that allows consistency and repetitively among projects and analysis of data. As added value, a well-designed metric helps not only to yield measures but also to plan change actions on the product/system attribute or capability. Finally, the process in the [8] strategy is poorly specified compared to that in GOCAME or SIQinU [14] strategies.

6 Conclusions

As the contributions mentioned in the Introduction Section, firstly, we have characterized and represented relevant Usability and UX features of mobile apps with regard to system, system-in-use and context entities. Secondly, we have analyzed Usability, Actual Usability and UX relationships regarding also EQ and QinU views, as well as specific Usability sub-characteristics and attributes for mobile apps in the light of a holistic evaluation approach. This evaluation approach is made up of a *quality modeling framework* (where 2Q2U is a subset) and *M&E strategies*, which in turn are based on three principles namely: a *M&E conceptual framework* (i.e. the C-INCAMI conceptual framework which is rooted in ontologies), *process view specifications*, and *method specifications*. So given the target information need, we can select the specific strategy and its tailored processes and methods in order to fulfill that specific purpose aimed at performing evaluations, analysis and recommendations. To this, we illustrated an evaluation study for the Facebook mobile app from the system Usability viewpoint, using the GOCAME strategy.

Of course, the Facebook study was made on the basis of a proof of concept as a typical social network app. But if we could have had control of the source code obviously GOCAME can be tailored to support change actions based on recommendations for improvement on weak performing indicators and re-evaluation of the new app version. An app with design features that jeopardize Effectiveness, Efficiency, Safety or Satisfaction (i.e. the do and be UX goals) can potentiate risks that it will not meet its business objectives. Evaluating these high level non-functional requirements such as Satisfaction, Actual Usability may feed back into detailed Usability, Functional and Information Quality, Security, etc. attributes and design requirements to maximize the quality of the user's experience and to minimize the likelihood of adverse consequences. Hence, our holistic evaluation approach can give support by means of specific strategies –as SIQinU- to the QinU/EQ/QinU improvement cycles. Ongoing research focuses on further utilizing our evaluation approach for QinU/EQ/QinU cycles for improving the design of mobile apps.

References

1. Apple iOS Human Interface Guidelines (2014), `http://developer.apple.com/library/ios#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html` (retrieved by January)
2. Bevan, N.: Extending Quality in Use to provide a Framework for Usability Measurement. In: Kurosu, M. (ed.) HCD 2009. LNCS, vol. 5619, pp. 13–22. Springer, Heidelberg (2009)

3. Bevan, N.: International Standards for Usability Should Be More Widely Used. Journal of Usability Studies 4(3), 106–113 (2009)
4. Dujmovic, J.: Continuous Preference Logic for System Evaluation. IEEE Transactions on Fuzzy Systems 15(6), 1082–1099 (2007)
5. Google User Interface Guidelines (2014), `http://developer.android.com/guide/practices/ui_guidelines/index.html` (retrieved by January)
6. Ham, D.-H., Heo, J., Fossick, P., Wong, W., Park, S., Song, C., Bradley, M.: Model-based Approaches to Quantifying the Usability of Mobile Phones. In: Jacko, J.A. (ed.) HCI 2007. LNCS, vol. 4551, pp. 288–297. Springer, Heidelberg (2007)
7. Hassenzahl, M.: User Experience: Towards an experiential perspective on product quality. In: 20th Int'l Conference of the Assoc. Francophone d'IHM, vol. 339, pp. 11–15 (2008)
8. Heo, J., Ham, D.-H., Park, S., Song, C., Chul, W.: A framework for evaluating the usability of mobile phones based on multi-level, hierarchical model of usability factors. Interacting with Computers 21(4), 263–275 (2009)
9. ISO/IEC 25010: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models (2011)
10. ISO/IEC 15939: Software Engineering - Software Measurement Process (2002)
11. ISO/IEC 9126-1: Software Engineering - Product Quality - Part 1: Quality Model (2001)
12. Lew, P., Olsina, L.: Relating User Experience with MobileApp Quality Evaluation and Design. In: Sheng, Q.Z., Kjeldskov, J. (eds.) ICWE Workshops 2013. LNCS, vol. 8295, pp. 253–268. Springer, Heidelberg (2013)
13. Lew, P., Qanber, A.M., Rafique, I., Wang, X., Olsina, L.: Using Web Quality Models and Questionnaires for Web Applications Evaluation. In: IEEE Proc., QUATIC, pp. 20–29 (2012)
14. Lew, P., Olsina, L., Becker, P., Zhang, L.: An Integrated Strategy to Systematically Understand and Manage Quality in Use for Web Applications. Requirements Engineering Journal 17(4), 299–330 (2012)
15. Nayebi, F., Desharnais, J.-M., Abran, A.: The state of the art of mobile application usability evaluation. In: 25th IEEE Canadian Conference on Electrical Computer Engineering, pp. 1–4 (2012)
16. Nielsen, J., Budiu, R.: Mobile Usability. New Riders, Berkeley (2012)
17. Olsina, L., Lew, P., Dieser, A., Rivera, B.: Updating Quality Models for Evaluating New Generation Web Applications. In: Abrahão, S., Cachero, C., Cappiello, C., Matera, M. (eds.) Journal of Web Engineering, Special Issue: Quality in New Generation Web Applications, vol. 11(3), pp. 209–246. Rinton Press, USA (2012)
18. Olsina, L., Papa, F., Molina, H.: How to Measure and Evaluate Web Applications in a Consistent Way. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) Web Engineering: Modeling and Implementing Web Applications. HCIS, pp. 385–420. Springer (2008)
19. Sohn, T., Li, K.A., Griswold, W.G., Holland, J.: A diary study of mobile information needs. In: ACM, Conference CHI 2008, Florence, Italy, pp. 433–442 (2008)

Finding Implicit Features in Consumer Reviews for Sentiment Analysis

Kim Schouten and Flavius Frasincar

Erasmus University Rotterdam
P.O. Box 1738, NL-3000 DR
Rotterdam, The Netherlands
{schouten,frasincar}@ese.eur.nl

Abstract. With the explosion of e-commerce shopping, customer reviews on the Web have become essential in the decision making process for consumers. Much of the research in this field focuses on explicit feature extraction and sentiment extraction. However, implicit feature extraction is a relatively new research field. Whereas previous works focused on finding the correct implicit feature in a sentence, given the fact that one is known to be present, this research aims at finding the right implicit feature without this pre-knowledge. Potential implicit features are assigned a score based on their co-occurrence frequencies with the words of a sentence, with the highest-scoring one being assigned to that sentence. To distinguish between sentences that have an implicit feature and the ones that do not, a threshold parameter is introduced, filtering out potential features whose score is too low. Using restaurant reviews and product reviews, the threshold-based approach improves the F_1-measure by 3.6 and 8.7 percentage points, respectively.

1 Introduction

With the explosion of online shopping at e-commerce companies like Amazon (US), Bol (NL), Alibaba (CN), etc., the use of consumer product reviews has become instrumental in the decision making process of consumers. In fact, potential consumers trust reviews from other consumers more than information on the vendor's website [1]. As a result, the number of reviews for a single product can be quite high, especially for a popular product. When a consumer is interested in the overall sentiment of a product, (s)he must first read through many of the reviews to come to a conclusion. Since reading through these reviews is a tedious process, this may hinder decision making. Therefore an efficient way of displaying the overall sentiment of a product based on costumer reviews is desirable.

Much of the current research in the analysis of product reviews is concerned with classifying the overall sentiment for a certain product. To better describe the overall sentiment of a product, it is useful to look at the sentiment per product aspect, from now on referred to as a feature. Sentiment classification per feature can be difficult as a customer review does not have a standard structure and may include spelling errors and synonyms for product features. Although a consumer might explicitly mention a feature for a product, many of the important features are mentioned implicitly as well. For example:

"The battery of this phone is quite good."
"The phone lasts all day."

In the first sentence, the battery is explicitly mentioned and the second one refers to the battery lasting all day. Notice that while in the second sentence the battery is not explicitly mentioned, we can infer that the comment is about the battery. This inference is based on the other words in the sentence that direct the reader towards the actual feature being described. This mapping from words in the sentence to the implied feature must be shared between writer and reader of a text in order for the reader to understand what the writer meant to imply. Because of this, it is usually a small group of well-known, coarse-grained features that is used implicitly. Examples include generic features like price, size, weight, etc., or very important product-specific features like the already mentioned battery, sound quality, ease of use, etc. Since it is this class of features that is often implied, it is important to include them in any sentiment analysis application, as they represent key features for consumers.

This research presents a method to both determine whether an implicit feature is present in a sentence, and if so, which one it is. After describing some of the related work that inspired this research, the method will be presented. Then, the two data sets that are used in the experiments are discussed, followed by the evaluation of the proposed method. This will lead to the conclusions and suggestions for future work in the last section.

2 Related Work

While many methods have been proposed to find features for the task of aspect-level sentiment analysis, most of them focus on explicit features only. This is logical, given that the vast majority of the features in consumer reviews is mentioned explicitly. However, as discussed in the previous section, it is often the important features that are mentioned implicitly. Alas, only few works focus on this task. One of the first to address the problem of detecting implicit features is [8]. An interesting solution is presented in the form of semantic association analysis based on Pointwise Mutual Information. However, since no quantitative results are given, it is impossible to know how well this method performs.

In [5], a method based on co-occurrence Association Rule Mining is proposed. It is making use of the co-occurrence counts between opinion words and explicit features. The latter can be extracted from labeled data, or can be provided by an existing method that finds explicit features. Association rule mining is used to create a mapping from the opinion words to possible features. The opinion word then functions as the antecedent and the feature as the consequent in the rules that are found. When an opinion word is encountered without a linked feature, the list of rules is checked to see which feature is most likely implied by that opinion word. On a custom set of Chinese mobile phone reviews, this method is reported to yield an F_1-measure of 74%.

Similar to [5], the same idea of association rule mining is used in [9]. With association rule mining being used to find a set of basic rules, three possible ways of extending the set of rules are investigated: adding substring rules, adding dependency rules, and adding constrained topic model rules. Especially the latter turned out to be a successful way of improving the results. By constraining the topic model (e.g., Latent Dirichlet Allocation [2] in this case), to include one of the feature words and build the topic around that word, meaningful clusters are generated. Thus, a different way of finding co-occurrences between features and other words in the text is used, and it is reported that this complements the association rule mining method. The best reported result is an F_1-measure of 75.51% on a Chinese data set of mobile phone reviews.

Instead of using annotated explicit features, [10] uses the idea of double propagation [7] to find a set of explicit words and a set of opinion words. An advantage is that the found explicit features are already linked to appropriate opinion words. Then a co-occurrence matrix is created, not between only opinion words and explicit features, but between the words in the sentences and the found explicit features. In this way, the right implicit feature is chosen, not based on just the opinion words in the sentence, but based on all words in the sentence. The opinion words in the sentence are used to constrain the number of possible features from which the right one must be chosen: only features that have co-occurred with the encountered opinion word before, are eligible to be chosen.

In the previously introduced method, for each eligible explicit feature, a score is computed that represents the average conditional probability of a feature being implied, given the set of words in the sentence. The feature with the highest score is chosen as the implicit feature for this sentence. This method is reported to yield an F_1-measure of 0.80 and 0.79 on a Chinese corpus of mobile phone reviews, and a Chinese collection of clothes reviews, respectively. Like [9], it uses all words to find implicit features instead of only opinion words as in [5], and, apart from a small seed set of opinion words, it operates completely unsupervised.

However, there are several drawbacks that are apparent, both in [5], [9], and in [10]. The first problem is that only features that have been found as explicit features somewhere in the corpus can be chosen as implicit features. This assumes that the same features are present in reviews, both explicitly and implicitly. However, as we have discussed before, well-known or important features are implied more often than features that are less important or less described. Furthermore, by counting the co-occurrence frequencies between a feature that is mentioned explicitly and the words in the sentence, it is assumed that when the feature is used implicitly, the same sentential context is present. We argue, however, that this is not necessarily the case. For example, when saying that ‘this phone is too expensive’, the word ‘expensive’ prevents the word ‘price’ from being used. Either one uses the word ‘expensive’, or one uses the word ‘price’. Because of that, there is no real co-occurrence between ‘expensive’ and ‘price’, even though the first definitely points to the latter as its implicit feature.

3 Method

In this section the issues discussed in the previous section are addressed and an algorithm is presented that improves upon previous work in the given, more realistic, scenario. This scenario entails the following:

– Sentences can have both explicit and implicit features;
– Sentences can have zero or more implicit features;
– Implicit features do not have to appear explicitly as well;
– The sentential context of explicit features does not have to be the same as the sentential context for implicit features.

The algorithm first scans the training data and constructs a list F of all unique implicit features, a list O of all unique lemmas (i.e., the syntactic root form of a word) and their frequencies, and a matrix C to store all co-occurrences between annotated implicit features and the words in a sentence. Hence, matrix C has dimensions $|F|$ x $|O|$.

When F, O, and C have been constructed, processing the test data goes as follows. For each potential implicit feature f_i, a score is computed that is the sum of the co-occurrence of each word in the sentence divided by the frequency of that word:

$$score_{f_i} = \frac{1}{v} \sum_{j=1}^{v} \frac{c_{i,j}}{o_j}, \tag{1}$$

where v is the number of words, f_i is the ith feature in F for which the *score* is computed, j represents the jth word in the sentence, $c_{i,j}$ is the co-occurrence frequency of feature i and lemma j in C, and o_j is the frequency of lemma o in O. Subsequently, for each sentence the highest scoring feature is chosen.

However, since there are many sentences without any implicit feature, a threshold is added, such that the highest scoring feature must exceed the threshold in order to be chosen. If the computed score does not exceed the threshold, the considered implicit feature is not assigned to that sentence. The pseudocode for the whole process is shown in Alg. 1, where the training process is shown (i.e., constructing co-occurrence matrix C and lists O and F), and in Alg. 2, where the processing of new sentences using the trained algorithm is shown.

The optimal threshold is computed based on the training data only, and consists of a simple linear search. A range of values is manually defined, all of them which are then tested consequently. The values ranged from 0 to 1, with a step size of 0.001. The best performing threshold is then used when evaluating on the test data. Since there is only one parameter to train and the range of possible values is rather limited, more advanced machine learning techniques were not deemed necessary to arrive at a good threshold value.

A limitation of this method is the fact that it will choose at most one implicit feature for each sentence. Both of our data sets, as can be seen in the next section, contain sentences that have more than one implicit feature. In these cases, chances are higher that the chosen implicit feature is in the golden standard,

Algorithm 1. Training the algorithm with annotated data

```
Initialize list of unique word lemmas with frequencies $O$
Initialize list of unique implicit features $F$
Initialize co-occurrence matrix $C$
for sentence $s \in$ training data do
    for word $w \in s$ do
        if $\neg(w \in O)$ then
            add $w$ to $O$
        end if
        $O(w) = O(w) + 1$
    end for
    for implicit feature $f \in s$ do
        if $\neg(f \in F)$ then
            add $f$ to $F$
        end if
        for word $w \in s$ do
            if $\neg((w, f) \in C)$ then
                add $(w, f)$ to $C$
            end if
            $C(w, f) = C(w, f) + 1$
        end for
    end for
    Determine optimal threshold.
end for
```

Algorithm 2. Executing the algorithm to process new sentences

```
for sentence $s \in$ test data do
    $currentBestFeature = empty$
    $scoreOfCurrentBestFeature = 0$
    for feature $f \in F$ do
        $score = 0$
        for word $w \in s$ do
            $score = score + C(w, f)/O(w)$
        end for
        if $score > scoreOfCurrentBestFeature$ then
            $currentBestFeature = f$
            $scoreOfCurrentBestFeature = score$
        end if
    end for
    if $scoreOfCurrentBestFeature > threshold$ then
        Assign $currentBestFeature$ to $s$ as its implicit feature
    end if
end for
```

but all features beyond the first will be missed by the algorithm. Another limitation is the obvious need for labeled data. Since this method is trained, not on explicit features, which can be determined by some other method, but on annotated implicit features, a sufficient amount of annotated data is required for our method to work properly.

4 Data Analysis

This section presents an overview of the two data sets that are used to train and evaluate the proposed method and its variants. The first data set is a collection of product reviews [6], where both explicit and implicit features are labeled. The second data set consists of restaurant reviews [4], where explicit aspects are labeled, as well as implicit aspect categories. Each sentence can have zero or more of these coarse-grained aspect categories. The restaurant set features five different aspect categories: 'food', 'service', 'ambience', 'price', and 'anecdotes/miscellaneous'. Since these aspects are implied by the sentence instead of being referred to explicitly, they function as implicit features as well. However, since there are only five options to choose from, it is much easier to obtain good performance on the restaurant set compared to the product set, where there are many different implicit features. Because of this, results for both data sets are not directly comparable. Even so, it is interesting to see how the proposed method performs on different data.

4.1 Product Reviews

The collection of product reviews are extracted from amazon.com, covering five different products: Apex AD2600 Progressive-scan DVD player, Canon G3, Creative Labs Nomad Jukebox Zen Xtra 40GB, Nikon Coolpix 4300, and Nokia 6610. Because the primary purpose of this data set is to perform aspect-level sentiment analysis, it is the case that features are only labeled as a feature when an opinion is expressed about that feature in the same sentence. In the example below, both sentences have a feature 'camera', but only in the second sentence is 'camera' labeled as a feature since only in the second sentence it is associated with a sentiment word.

"I took a picture with my phone's *camera*."
"The *camera* on this phone takes *great* pictures."

Because the product data set contains a lot of different, but sometimes similar, features, a manual clustering step has been performed. This makes the set of features more uniform and reduces unnecessary differences between similar features. It also removes some misspellings that were present in the data set. In total, the number of unique implicit features is reduced from 47 to 25.

As can be seen in Fig. 1, there are not many sentences with an implicit feature. This only stresses the need for a good selection criterion to distinguish the ones with an implicit feature from the ones that do not have one. There is also a small number of sentences (0.2%) that have two implicit features. Since the algorithm will only choose zero or one implicit feature for each sentence, this can potentially impact performance in a negative way. The second implicit feature will always be missed, leading to a lower recall. This is however slightly mitigated by the fact that it is easier to pick a correct feature, as it is checked against both annotated features in the sentence.

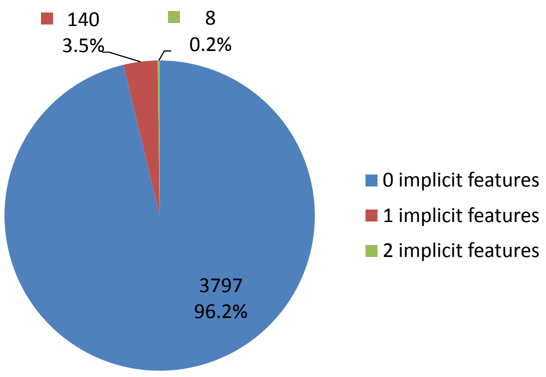


Fig. 1. Distribution of sentences in the product review data set, according to the number of implicit features they contain

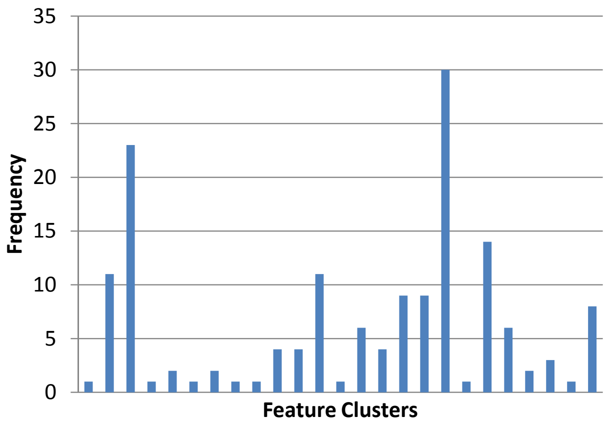


Fig. 2. Frequencies for all 25 unique feature clusters in the product review data set

In Fig. 2, the frequency distribution of the set of implicit features is given. Frequency is measured as the number of sentences a certain implicit feature appears in. As can be seen, there are quite a few implicit features which appear in only a couple of sentences. Fifteen out of the 25 feature appear in less than 5 sentences, with eight features occurring in only one sentence. This makes it extremely difficult to learn a classifier that is able to find these features. In case of the features that appear only once, it is completely impossible to devise a classifier, since they cannot both appear in the test and in the training set.

4.2 Restaurant Reviews

Compared to the product reviews, the restaurant review data set has clearly different statistical characteristics, as shown in Fig. 3. Where the product review set has only a few sentences that contain an implicit feature, in the restaurant set, all of them have an aspect category, which we will regard as an implicit feature in this research. The much bigger size, together with the already mentioned fact that there are only five different implicit features in this data set, makes for a much easier task. To measure the influence of the threshold parameter, the fifth category of 'anecdotes/miscellaneous' is removed from the data set. Since this category does not really describe a concrete implicit feature, removing it leaves us with sentences that do not have any implicit feature, allowing the performance of the threshold to be assessed on this data as well.

Compared to the product reviews data set, the frequency distribution of the implicit features in the restaurant reviews set, shown in Fig. 4 is more balanced. Every features has at least a couple of hundred sentences in which it is appearing. The one outlier is the 'food' category, which appears twice as much as the second largest feature which is 'service'. Still, the difference between the feature that appears the most ('food') and the one that appears the least ('price') is only a factor of three, whereas for the product features, this would be much higher (i.e., around 30).

5 Evaluation

All evaluations are performed using 10-fold cross-evaluation. Each tenth of the data set is used to evaluate an instance of the algorithm that is trained on the other 90% of the data. Both the co-occurrence frequencies and the threshold parameter are determined based on the training data only. When evaluating the algorithm's output, the following definitions are used:

– $truePositives$ are the features that have been correctly identified by the algorithm;
– $falsePositives$ are those features that have been annotated by the algorithm, that are not present in the golden standard;
– $falseNegatives$ are those features that are present in the golden standard, but that have not been annotated by the algorithm;
– $trueNegatives$ are features that are not present in the golden standard, and are correctly not annotated by the algorithm.

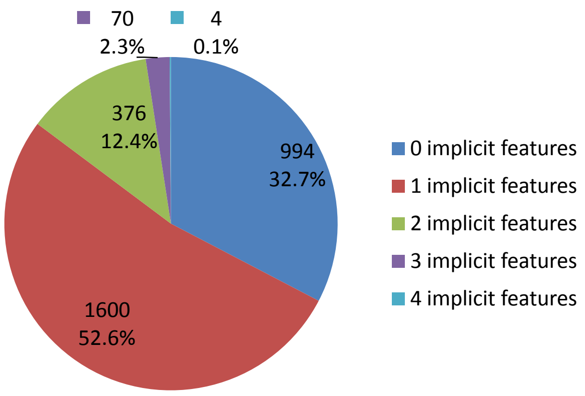


Fig. 3. Distribution of sentences in the restaurant review data set, according to the number of implicit features they contain

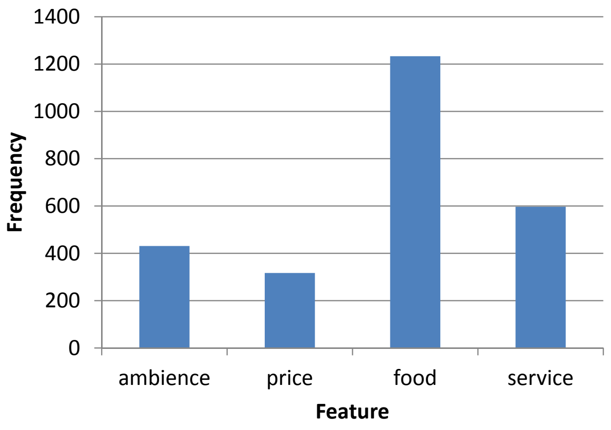


Fig. 4. Frequencies for all 4 unique features in the restaurant review data set

When evaluating, a feature always has to be the same one as in the golden feature to count as a true positive. Simply stating that there is some implicit feature in a sentence, which might be true, is not enough. In order to count as a true positive, it has to be the right implicit feature. From this follows that, given a sentence with only one annotated implicit feature and one golden implicit feature, when the algorithm correctly identifies that a sentence contains an implicit feature, but it chooses the wrong one, the wrongly assigned feature will count as a false positive and the annotated one will count as a false negative. As such, both precision and recall will be lower. In general the algorithm can make three kinds of mistakes:

- State that a sentence contains an implicit feature, while actually it does not: precision will be lower;
- State that a sentence does not contain an implicit feature, while actually it does: recall will be lower;
- Correctly stating that a sentence contains an implicit feature, but picking the wrong one: both precision and recall will be lower.

Because of the ten-fold cross-validation, the reported scores are computed on the sum of the ten confusion matrices (i.e., derived from the ten folds). For example, precision would be computed as:

$$precision = \frac{\sum_{fold=1}^{10} truePositives_{fold}}{\sum_{fold=1}^{10} truePositives_{fold} + falsePositives_{fold}}. \tag{2}$$

Recall is computed in a similar way, leaving the F$_1$-measure, being the harmonic mean of precision and recall, to be computed as usual. In the end, each sentence will be processed exactly once, but will be used nine times as training instance.

The proposed algorithm is tested both with and without the proposed threshold, to assess the benefit of training such a threshold. Furthermore, both versions are evaluated using a Part-of-Speech filter. The latter is used to filter out words in the co-occurrence matrix that may not be useful to find implicit features. Besides evaluating using all words (i.e., including stopwords), both algorithms are evaluated using an exhaustive combination of four word groups, namely nouns, verbs, adjectives, and adverbs.

Since the algorithm without a threshold will generally choose some implicit feature for every sentence, any trained threshold is expected to surpass that score. To provide more insight in this problem, a maximum score is also provided. This maximum score is computed by filtering out all sentences without any implicit feature and then letting the algorithm simply pick the most appropriate feature. This situation reflects a perfect threshold that is always able to make the distinction between the presence or absence of an implicit feature. Obviously, in reality, the trained threshold does not come close to this ideal performance, but including this ideal line allows the separation of errors due to threshold problems from errors due to not picking the right feature. The latter is an intrinsic problem of the algorithm, not of the threshold. With this in mind, one can see that the gap between the ideal line and the bars represents errors that can be attributed to the threshold, while the gap between 100% performance and the ideal line represents errors that can be attributed to the method of using co-occurrence frequencies to find the right feature.

The results on the product review data set are presented in Fig. 5, whereas the results on the restaurant review data set are presented in Fig. 6. In each graph there are two grouped bars for each Part-of-Speech filter, where the first bar shows the performance without a threshold and the second bar the performance with the trained threshold. The line above the bars represents the ideal, or maximum possible, performance with respect to the threshold, as discussed above. There are 16 different Part-of-Speech filters shown in both graphs. The first `all`, simply means that all words, including stopwords, are used in the co-occurrence matrix. The other fifteen filters only allow words of the types that are mentioned, where NN stands for nouns, VB stands for verbs, JJ stands for adjectives, and RB stands for adverbs.

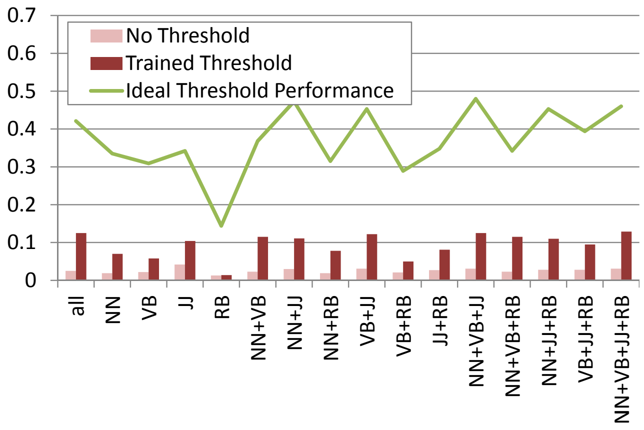


Fig. 5. The performance on the product review data set in F_1-measure for the various PoS-filters

For the product set, it is beneficial to keep as many words as possible, something that is probably caused by the small size of the data set. However, removing stopwords results in a slightly higher performance: the `NN+VB+JJ+RB` filter scores highest. Looking at the four individual categories, it is clear that adjectives are

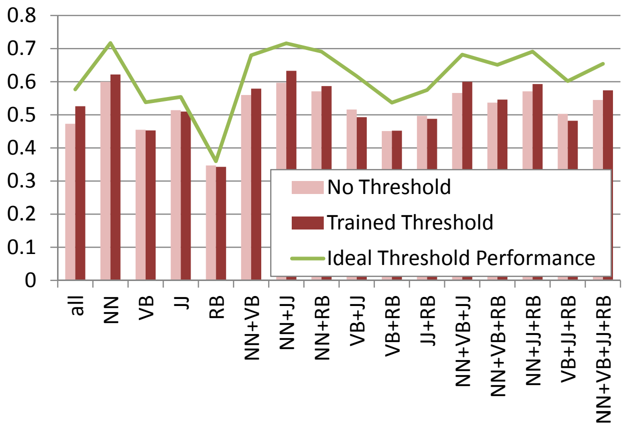


Fig. 6. The performance on the restaurant review data set in F_1-measure for the various PoS-filters

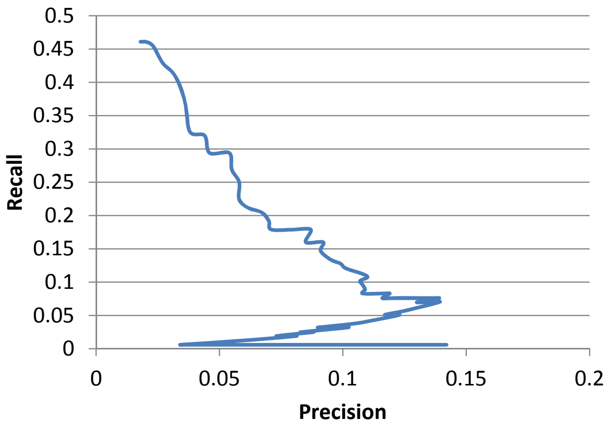


Fig. 7. The precision-recall trade-off on the product review data set, when manipulating the threshold variable (using the `NN+VB+JJ+RB` filter)

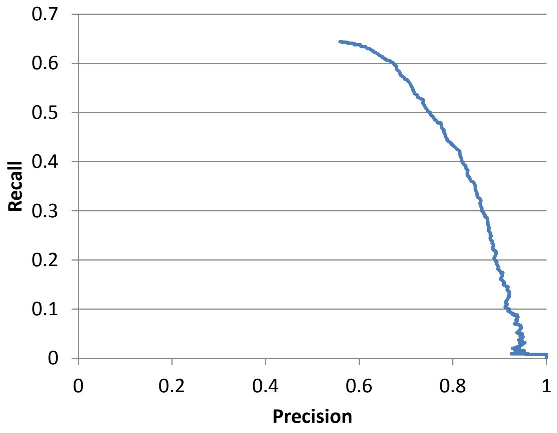


Fig. 8. The precision-recall trade-off on the restaurant review data set, when manipulating the threshold variable (using the `NN+JJ` filter)

most important to find implicit features. For the restaurant set, the situation is a bit different. Here, nouns are the most important word group, followed by adjectives. Because of its larger size, it is possible to remove verbs and adverbs without any detrimental effects. Hence, the `NN+JJ` filter yields the best performance.

Another observation we can draw from comparing Fig. 5 and Fig. 6 is that the restaurant set is in general much easier for the algorithm to process. Not only are the ideal performances higher on the restaurant set, also the gap between the ideal and the realized performance is smaller. The most likely reason for this

Table 1. Comparison of results with Zhang & Zhu [10], with and without the proposed threshold. Reported scores are F$_1$-measures for the best scoring Part-of-Speech filter. Differences between scores are expressed in percentage points (pp.), the arithmetic difference between two percentages.

product review data set

method	no threshold	trained threshold	difference
Zhang & Zhu	1.2% (`all`)	1.4% (`NN+VB+JJ+RB`)	+0.2 pp.
proposed method	4.2% (`JJ`)	12.9% (`NN+VB+JJ+RB`)	+8.7 pp.
difference	+3 pp.	+11.5 pp.	

restaurant review data set

method	no threshold	trained threshold	difference
Zhang & Zhu	31.5% (`all`)	32.4% (`all`)	+0.9 pp.
proposed method	59.7% (`NN+JJ`)	63.3% (`NN+JJ`)	+3.6 pp.
difference	+28.2 pp.	31.1 pp.	

difference is the fact that in the restaurant set there are roughly 2000 sentences that contain at least one of the four possible implicit features, whereas in the product set, there are 140 sentences that contain at least one of 25 possible implicit features. Not only does this render the task of picking the right feature more difficult, it also increases the complexity of judging whether a sentence contains one of these features.

The fact that the vast majority of the product set has no implicit feature at all makes the utilization of a threshold all the more important. This is in contrast to the restaurant set, where two-thirds of the sentences have an implicit feature. Again, this is shown clearly in Fig 5 and Fig 6: the relative improvement of the threshold is much higher for the product data than the restaurant data.

In Fig. 7 and Fig. 8, the precision-recall trade-off is shown for the best scoring Part-of-Speech filter. The restaurant set yields a well-defined curve, which is to be expected due to the large quantity of available data. Note that as in all other graphs, two tasks are being evaluated: determine whether or not there is an implicit feature in a sentence, and if so, determine which one it is. This is the reason that, even with a threshold of zero, the recall will not be 100%: while it does state that every sentence will have an implicit feature, it still has to pick the right one in order to avoid a lower recall (and precision for that matter).

A comparison with the method of Zhang & Zhu [10] is given in Table 1. To increase comparability, both methods are tested with all sixteen possible Part-of-Speech filters (only the best one is reported). To be fair, the original method is also tested with a threshold added, using the same procedure as for the proposed method, even though this contributes little to its performance.

Interestingly, the effect of the threshold is bigger on the product set compared to the restaurant set. This might point to the fact that training this parameter can partly mitigate the negative effect of having a small data set. Consequently, when the data set is larger, the algorithm on its own already performs quite well, leaving less room for improvement by other methods, like adding a threshold.

6 Conclusion

Based on the diagnosed shortcomings in previous work, we proposed a method that directly maps between implicit features and words in a sentence. While the method effectively becomes a supervised one, it is not flawed in its assumptions as previous work, and performance is reported to increase on the two used data sets. Furthermore, a more realistic scenario is implemented wherein the proposed method not only has to determine the right implicit feature, but also whether one is actually present or not.

The proposed algorithm shows a clear improvement with respect to an existing algorithm on the two data sets considered, as it is better in distinguishing between sentences that have an implicit feature and the ones that do not. Both for product reviews and restaurant reviews, the same general improvement is observed when implementing this threshold, even though the actual performance differs much between the two data sets.

Analysis of the performance of the algorithm in relation to the characteristics of the two data sets clearly shows that having less data, but more unique implicit features to detect severely decreases performance. While the proposed algorithm is much better in dealing with this lack of data, the results for that particular data set are still too low to be useful in practice. On the set of restaurant reviews, being of adequate size and having only four unique implicit features, the proposed algorithm yields promising results. Adding a threshold further boosts the performance by another 3 percentage points, which is highly desirable for this kind of user generated content.

A primary suggestion for future work is to learn a threshold for each individual implicit feature, instead of one general threshold that applies to all implicit features. We hypothesize that because some features are used more often in an implicit way than others, and the sentential context differs from feature to feature as well, it makes sense to learn a different threshold for each unique implicit feature.

Also interesting could be to adjust the algorithm to be able to choose more than one implicit feature. Especially on the restaurant set, where about 14% of the sentences have more than one implicit feature, performance could be improved. Possible ways of doing this include choosing all features whose score exceeds the threshold, or employ a classifier that determines how many implicit features are likely to be present. The latter could also be investigated as a possible alternative for the threshold.

Last, a move from word based methods, like this one, toward concept-based methods, as advocated in [3], would be interesting as well. For example, cases like:

"This phone doesn't fit in my pocket."

is very hard to process based on words alone. It is probably feasible to determine that the implicit feature here is 'size', if enough training data is at hand, but determining that this sentence represents a negative sentiment, since mobile phones are *supposed* to fit in ones pocket, seems extremely hard for word-based methods. While concept level methods are still in their infancy, they might be up to this challenge, since common sense knowledge, world knowledge, and domain knowledge are integrated in such an approach.

Acknowledgments. The authors are partially supported by the Dutch national program COMMIT. We also would like to thank Andrew Hagens, Gino Mangnoesing, Lotte Snoek, and Arno de Wolf for their contributions to this research.

References

1. Bickart, B., Schindler, R.M.: Internet Forums as Influential Sources of Consumer Information. Journal of Interactive Marketing 15, 31–40 (2001)
2. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent Dirichlet Allocation. Journal of Machine Learning Research 3, 993–1022 (2003)
3. Cambria, E., Schuller, B., Xia, Y., Havasi, C.: New Avenues in Opinion Mining and Sentiment Analysis. IEEE Intelligent Systems 28, 15–21 (2013)
4. Ganu, G., Elhadad, N., Marian, A.: Beyond the Stars: Improving Rating Predictions using Review Content. In: Proceedings of the 12th International Workshop on the Web and Databases, WebDB 2009 (2009)
5. Hai, Z., Chang, K., Kim, J.: Implicit Feature Identification via Co-occurrence Association Rule Mining. In: Gelbukh, A.F. (ed.) CICLing 2011, Part I. LNCS, vol. 6608, pp. 393–404. Springer, Heidelberg (2011)
6. Hu, M., Liu, B.: Mining and Summarizing Customer Reviews. In: Proceedings of 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2004), pp. 168–177. ACM (2004)
7. Qiu, G., Liu, B., Bu, J., Chen, C.: Opinion Word Expansion and Target Extraction through Double Propagation. Computational Linguistics 37, 9–27 (2011)
8. Su, Q., Xu, X., Guo, H., Guo, Z., Wu, X., Zhang, X., Swen, B., Su, Z.: Hidden Sentiment Association in Chinese Web Opinion Mining. In: Proceedings of the 17th International Conference on World Wide Web (WWW 2008), pp. 959–968. ACM (2008)
9. Wang, W., Xu, H., Wan, W.: Implicit Feature Identification via Hybrid Association Rule Mining. Expert Systems with Applications 40, 3518–3531 (2013)
10. Zhang, Y., Zhu, W.: Extracting Implicit Features in Online Customer Reviews for Opinion Mining. In: Proceedings of the 22nd International Conference on World Wide Web Companion (WWW 2013 Companion), pp. 103–104. International World Wide Web Conferences Steering Committee (2013)

From Choreographed to Hybrid User Interface Mashups: A Generic Transformation Approach

Alexey Tschudnowsky[1], Stefan Pietschmann[2], Matthias Niederhausen[2], Michael Hertel[1], and Martin Gaedke[1]

[1] Technische Universität Chemnitz, Germany
{alexey.tschudnowsky,michael.hertel,gaedke}@informatik.tu-chemnitz.de
[2] T-Systems MMS, Germany
{stefan.pietschmann,matthias.niederhausen}@tu-dresden.de

Abstract. Inter-widget communication (IWC) becomes an increasingly important topic in the field of user interface mashups. Recent research has focused on so-called choreographed IWC approaches that enable self-organization of the aggregated components based on their messaging capabilities. Though a manual configuration of communication paths is not required anymore, such solutions bear several problems related to awareness and control of the emerging message flow. This paper presents a systematic approach to tackle these problems in the context of *hybrid* user interface mashups. We show how users can be made aware of the emerged IWC configuration and how they can adjust it to their needs. A reference architecture for development of hybrid mashup platforms, is derived and one implementation based on the publish-subscribe choreography model is given. We report on the results of a first user study and outline directions for the future research.

Keywords: inter-widget communication, user interface mashup, widgets, end-user development.

1 Introduction

User interface (UI) mashups have become a popular approach for end-user development. Based on autonomous but cooperative visual components called widgets, they promise to significantly lower the barrier for Web application development [2,19,10]. The development process of UI mashups usually implies three steps: First, finding appropriate widgets for composition; Second, placement and configuration of widgets on a common canvas; and finally, configuration of the cooperative behaviour by means of inter-widget communication (IWC). IWC hereby refers to the process of exchanging data between widgets and can be used to synchronize internal states of the aggregated components.

The IWC behaviour of a mashup can be defined either explicitly by a mashup designer (*orchestrated mashups*), emerge from the capabilities of the integrated components (*choreographed mashups*) or be defined by a combination of both (*hybrid mashups*) [22]. While orchestrated approaches aim at providing flexibility during mashup development by enabling designers to define the desired data flow manually, choreographed and hybrid solutions focus on keeping the development process lean and fast. In choreographed solutions, widgets “decide” autonomously on how to communicate and with whom. Hybrid mashups behave as choreographed ones, but provide additional means to restrict the emerging communication. EDYRA [19], DashMash [3] or EzWeb [13] are some approaches that exemplify orchestrated mashups. Using the ”wiring” metaphor, they enable mashup designers to connect ”inputs” and ”outputs” of components and, thus, specify data flow in a mashup. The target group of such platforms are skilled users and hobby developers, who are experienced with the concepts of operations, input/output parameters and data structures. OMELETTE [4], ROLE [10] and Open Application [8] projects follow the choreographed approach. Widgets communicate without a prior configuration - using the publish-subscribe messaging pattern each widget decides autonomously on which messages to send and which messages to receive. Chrooma+ [12] is an example of a hybrid platform. While widgets publish and subscribe for messages on their own, a mashup designer is still able to ”isolate” one or more widgets from their environment. The target group of choreographed and hybrid platforms are end-users, who have little to no programming skills but are experts in their corresponding business domains.

Though choreographed and hybrid mashups are considered to be more ”end-user-friendly”[8], they also pose some challenges with regard to awareness and control of what is happening in a mashup [21,8]. The major awareness problem caused by implicitly defined IWC is that users do not know which pairs of widgets *could* and which actually *do* communicate. Users have to learn the data and control flows as they use and explore the mashup. While in general this may merely frustrate users, such ”exploratory” interaction can also accidentally affect live data, causing undesired side effects. The major control problem is that, being defined implicitly and not as first-level concepts, communication paths cannot be blocked, modified or added directly by end-users. Possible reasons for intervention are, e.g., untrusted widgets or unexpected or undesired state synchronisations. A detailed analysis of these and other problems related to awareness and control in UI mashups can be found in our prior work in [5].

This paper presents an approach for systematic development of hybrid mashup platforms with IWC awareness and control in mind. The goal is to support end-user development of UI mashups by combining advantages of choreographed and orchestrated mashup platforms - self-emerging IWC with flexible visualization and tailoring facilities. In summary, the contributions of this paper are as follows:

– *A generic IWC model including corresponding visualization and tailoring mechanisms.* The model and mechanisms are used to communicate IWC behaviour to end-users and facilitate its configuration.
– *A reference architecture for hybrid mashup platforms.* The architecture enables systematic development of hybrid mashup platforms - both from scratch and by extension of existing choreographed ones.
– *Evaluation of the proposed concepts with end-users.* Experiments with 27 end-users assessed the efficiency and usability of the proposed mechanisms.

The rest of the paper is structured as follows. Section 2 gives a background on choreographed UI mashups and presents the steps required for systematic development of hybrid mashup platforms. Section 3 demonstrates one implementation of the proposed architecture for the publish-subscribe choreography strategy. An evaluation of the proposed awareness and control facilities is given in section 4. Finally, section 6 concludes the paper and derives directions for future research.

2 From Choreographed to Hybrid UI Mashups

The section presents a systematic approach to build hybrid mashup platforms. As the working principle of hybrid mashups is close to choreographed ones, the idea is to leverage existing choreographed platforms and to extend them with the missing visualization and tailoring functionality. The approach can also be used to build hybrid platforms from scratch.

In the following, different types of choreographed mashups with regard to utilized IWC models are presented. Afterwards, a unified communication model for visualization and tailoring of data flow is described. Based on the model, several awareness and control mechanisms are proposed. Finally, a reference architecture to support these mechanisms is given.

2.1 Choreographed UI Mashups

Choreographed UI mashups do not require mashup designers to specify data flow in a mashup. Instead, communication emerges in a self-organizing fashion depending on the messaging capabilities of widgets. Technically, different IWC strategies exist to enable "self-organization" of autonomous but cooperative components: message passing, publish-subscribe, remote procedure calls (RPC) and shared memory[24]. *Message passing* considers widgets as senders and recipients of structured application-specific data. Delivery can take place in uni-, multi- or broadcast fashion. *RPC* solutions enable widgets to offer operations, which can be invoked by others in synchronous or asynchronous way. Discovery of available operations happens either at widget design-time or at run-time, e.g., by means of centralized widget registries. In *publish-subscribe* systems, widgets emit and receive messages on different channels, also called "topics". The decision, which channels to publish or subscribe on, is met by widgets autonomously without intervention of a mashup designer. Finally, *shared memory* solutions enable widgets to read and to write to a common data space and, thus, autonomously exchange data among each other. Independently of the concrete communication model, widgets can communicate either directly or by means of the platform middleware. Platform-mediated communication enables loose coupling of aggregated components as well as additional services such as traffic monitoring and management. As to the authors' knowledge, all of the current UI mashups make use of platform middleware to implement IWC. The presented strategies offer techniques for implementation of self-organizing widget compositions. User awareness and control are out of their scope and thus require additional engineering on the side of the mashup development/execution platform.

2.2 Communication Model

The goal of the unified communication model presented here is to provide a common data structure for visualization and control mechanisms. The assumption for the definition of the model is that – from an end-users' point of view – widgets communicate in pairs by means of unidirectional message transfers. In terms of a concrete choreography model, messages have different semantics, e. g., invocation of a remote procedure or publication/subscription to some topic. Regardless of the model, the user-perceived result is that one widget receives data from another one. These considerations build a basis for the unified communication model described by the following data structure:

The unified communication model M is a graph $G = (V, E)$ with

- $V = \{v | v = (id, s)\}$ set of vertices with identifier id and state $s \in \{ENABLED, ISOLATED\}$. Each vertex corresponds to exactly one widget in a mashup.
- $E = \{e | e = (v_1, v_2, s, t)\}$ set of edges corresponding to possible communication paths between widgets corresponding to $v_1, v_2 \in V$ with state $s \in \{ENABLED, BLOCKED\}$ and with label t.

For all of the presented choreography models it is possible to define an algorithm which yields a unified communication model M. Section 3.2 presents one possible algorithm for publish-subscribe-based choreography models.

A data flow restricted by the unified model M takes place as follows:

- A widget corresponding to the vertex v is allowed to emit or receive messages only if the state s of the vertex v is $ENABLED$.
- A message m from a widget corresponding to v_1 is allowed to be delivered to a widget corresponding to v_2 only if $\exists e \in E : e = (v_1, v_2, ENABLED, t)$.
- The data flow takes place according to the utilized choreography strategy if none of the above restrictions apply.

2.3 Visualization and Tailoring Facilities

The following visualization and tailoring facilities are proposed to make mashup designers aware of the data flow in a UI mashup and to enable them to adjust it:

- States $s \in \{ENABLED, ISOLATED\}$ of vertices are visualized using borders of different color and type around the corresponding widgets.
- Potential communication paths $e = (v_1, v_2, s, t) \in E$ are visualized using arrows between widgets corresponding to v_1 and v_2. The arrow style indicates the state of the communication path $s \in \{ENABLED, BLOCKED\}$. Annotation t is displayed above the corresponding arrow to provide additional information on the communication path.

- Flashing icons on widget borders show which widgets corresponding to v_1 and v_2 are currently communicating along a communication path $e = (v_1, v_2, s, t) \in E$.
- For every vertex v, visualization of its state $s \in \{ENABLED, ISOLATED\}$ and in-/outgoing edges $e = (v, *) \in E$ can be turned on or off to avoid cognitive overload in case of strong connectivity.
- For every vertex v, its state $s \in \{ENABLED, ISOLATED\}$ can be toggled using corresponding user interface controls.
- For each edge $e = (v_1, v_2)$, its state $s \in \{ENABLED, BLOCKED\}$ can be toggled by clicking on the corresponding arrows.

2.4 Reference Architecture

The reference architecture (cf. Figure 1) acts as a blueprint for the development of awareness- and control-enabled hybrid mashup platforms. Some of the platform components (such as the *Awareness and Control Module* and the *Communication Model*) are independent of a chosen choreography model, whereas others (*Widget and Mashup Descriptors*, *Model Importer*, *Model Exporter* and *Message Broker*) are choreography-model-specific.

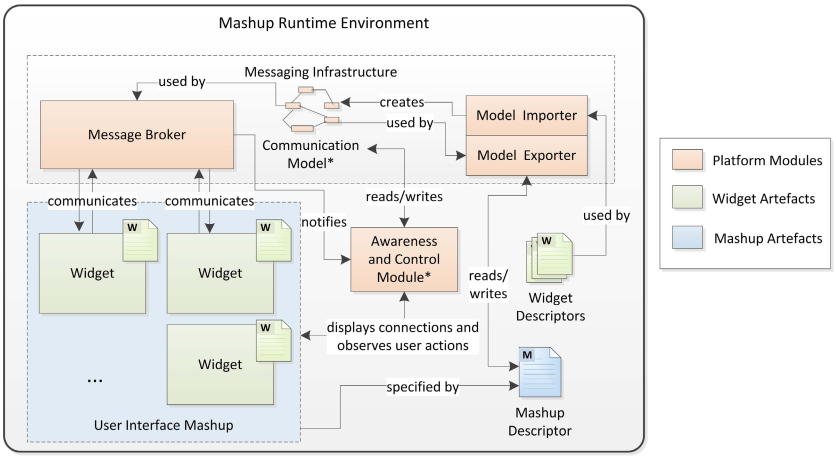


Fig. 1. Reference architecture for implementation of hybrid UI mashups. * - implementation can be shared between platforms with different choreography strategies.

The components of the reference architecture provide resources and services required for UI mashup development. *Widget Descriptors* describe IWC capabilities and default configuration parameters of installed widgets. Mashup layout, aggregated widget instances, user preferences and IWC configuration are specified by the *Mashup Descriptor*. All artefacts are dependent on the selected choreography model, but can be used by the *Model Importer* to derive a unified *Communication Model*. The *Awareness and Control Module* displays the model in the composition canvas according to the rules described in 2.3. It is also responsible for updating the *Communication Model* upon changes triggered by users, e. g., changing state of an edge upon clicks on the corresponding arrow. The *Message Broker* is the platform's communication middleware. Its goal is, first, to provide messaging functionality according to the rules of the underlying choreography approach and, second, to assure that restrictions of the unified *Communication Model* such as isolated widgets or blocked communication paths are respected. The *Awareness and Control Module* gets notified about activities within the *Message Broker* and displays activated or blocked communication paths to mashup users.

In the following, we present one example implementation of the reference architecture. It reuses an existing publish-subscribe-based UI mashup platform and extends it towards the missing awareness and control functionality. We selected a platform based on the publish-subscribe strategy because of the wide use of this approach in current choreography platforms.

3 Hybrid Mashups Based on Publish-Subscribe Choreography Model

Publish-subscribe-based IWC is a part of many choreographed UI mashup platforms [4,12,10]. Frameworks like OpenAjaxHub[1] or AmplifyJS[2] simplify the integration of the corresponding infrastructure into Web-based applications. The underlying communication strategy is, however, always the same and can be formalized as follows:

Let $m = (W, T)$ be a widget mashup with

- Reference ontology $T = \{t_n : t_n = (name_n, TYPE_n)\}$ being a set of concepts and associated message types $TYPE_n = \{value_n\}$
- Widget set $W = \{w_j : w_j = (id_j, PUB_j, SUB_j)\}$ with unique identifier id_j, set of publications $PUB_j = \{p_{jl} : p_{jl} \in T\}$ and set of subscriptions $SUB_j = \{s_{jk} : s_{jk} \in T\}$

Let $a = (wsender, t, data)$ with $wsender \in W, t \in wsender.PUB, data \in t.TYPE$ be a message emitted by a widget $wsender$. The message a is delivered to all widgets $w_i \in W : t \in w_i.SUB$.

The following implementation of a publish-subscribe-based hybrid mashup platform is based on two open-source projects - Apache Wookie[3] and Apache Rave[4]. Apache Wookie is a widget container for hosting W3C widgets[5], which are

[1] http://www.openajax.org/member/wiki/OpenAjax_Hub_2.0_Specification
[2] http://amplifyjs.com
[3] http://wookie.apache.org
[4] http://rave.apache.org
[5] http://www.w3.org/TR/widgets

stand-alone Web applications packaged for distribution on the Web. Apache Rave is a widget mashup environment, which enables aggregation of both W3C and OpenSocial[6] widgets on one canvas and provides a publish-subscribe messaging infrastructure for communication between the widgets. In the following, those parts of the projects are described in detail, which were extended towards the reference architecture.

3.1 Widget Descriptors

According to the W3C specification, widget metadata contains only basic information on the packaged application such as title, description, author, license etc. Currently there is no standard way to describe widget communication capabilities. However, this is essential for the proposed awareness and control mechanisms (cf. section 2.2). Therefore, an extension of the W3C metadata file is proposed to describe publications and subscriptions of widgets together with used topics. Listing 1.1 shows the proposed extension for a fictitious *Contacts* widget, which maintains a list of structured contact entries. An incoming phone number (topic *http://example.org/phoneNumber*) causes the widget to filter the list towards a contact with the passed number. If a contact is selected by user, its complete data is published on the *http://example.org/contact* topic. The schema of the involved messages is given in the *oa:topics* section. Both Apache Wookie and Rave were extended to support the extended widget metadata.

Listing 1.1. Extension of the W3C metadata file

```
<feature name="http://www.openajax.org/hub"
xmlns:oa="http://www.openajax.org/hub">

<!-- Declaration of topics -->
<oa:topics>
  <oa:topic oa:name="http://example.org/contact">
    <oa:schema oa:schemaType="JSON">
    <![CDATA[
    {"description":"A person",
      "type":"object",
      "properties":{
        "name":{"type":"string"},
        "age" :{
          "type":"integer",
          "maximum":125 }
    }}]]>
    </oa:schema>
  </oa:topic>

  <oa:topic oa:name="http://example.org/phoneNumber">
```

[6] `https://developers.google.com/gadgets/docs/dev_guide?csw=1`

```
    <oa:schema oa:schemaType="XML" oa:simpleType="xs:string"/>
  </oa:topic>

  <oa:topic ...>
</oa:topics>

<!-- Declaration of publications -->
<oa:publications>

<oa:publication oa:topicRef="http://example.org/contact"/>
  <oa:publication ...>
</oa:publications>

<!-- Declaration of subscribtions -->
<oa:subscriptions>
  <oa:subscription oa:topicRef="http://example.org/phoneNumber"/>

  <oa:subscription ...>
</oa:subscriptions>

</feature>
```

3.2 Model Importer

The module has been added to Apache Rave and is responsible for construction of the *Communication Model* based on *Widget Descriptors* and *Mashup Descriptors*. The construction of the model for publish-subscribe-based UI mashups is specified by the Algorithm 1.

Algorithm 1. Creating a Unified Communication Model for Publish-Subscribe-based UI Mashups

```
Input  : Publish-subscribe-based UI mashup $m = (W, T)$ as defined above
Output: Communication model $G = (V, E)$ as defined in section 2.2
 1. Set $V = \emptyset$, $E = \emptyset$
 2. For each widget $w_i \in W$, create a new vertex $v_i$ and set $V = V \cup v_i$
 3. For each pair of widgets $(w_i, w_j)$ and for each $p_{ik} \in w_i.PUB : p_{ik} \in w_j.SUB$,
    create a new edge $e_{ik} = (v_i, v_j, ENABLED, p_{ik})$ and set $E = E \cup e_{ik}$
 4. Return $G = (V, E)$
```

The resulting communication model reflects potential message flows in a publish-subscribe-based UI mashup. By default, all communication paths and widgets are in the $ENABLED$ state.

3.3 Awareness and Control Module

The *Awareness and Control Module* weaves the *Communication Model* into the composition canvas and updates it, based on user actions such as widget isolations or path blockades. It also highlights communicating partners according to notifications from the *Message Broker*. Visualization of the model takes place as proposed in Section 2.3 and is implemented using the jsPlumb[7] JavaScript drawing library (cf. Figure 2).

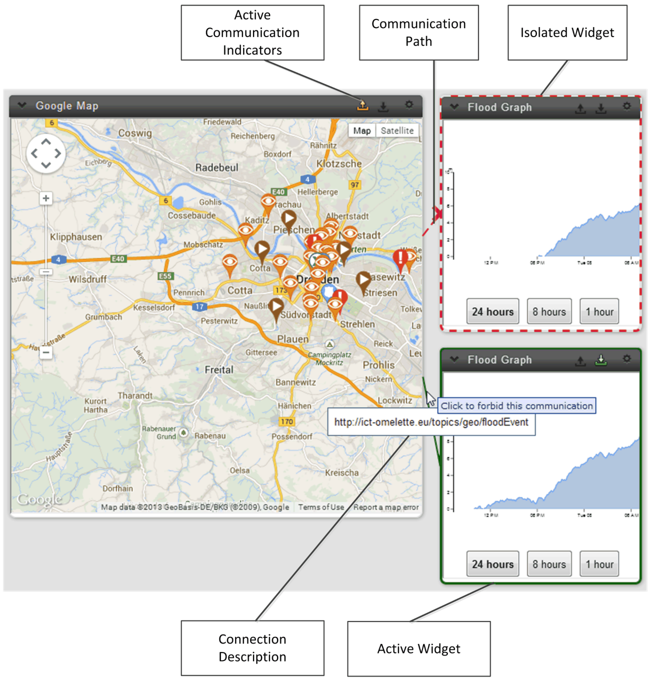


Fig. 2. Awareness and Control Mechanisms integrated into Apache Rave

3.4 Message Broker

The *Message Broker* is an existing component in the Apache Rave platform and is implemented using the OpenAjaxHub framework. It is responsible for routing messages between widgets according to the publish-subscribe strategy as specified above. The component has been extended to take the *Communication Model* into account while routing messages. The routing algorithm was refined as follows: a message $a = (wsender, t, data)$ from widget $wsender$ is allowed to be delivered to $\forall w_i \in W : t \in w_i.SUB$ if and only if $v_{wsender}.s = ENABLED \wedge v_{w_i}.s = ENABLED \wedge e.s = ENABLED : e = (v_{wsender}, v_{w_i})$.

[7] `http://jsplumbtoolkit.com/`

3.5 Model Exporter

The *Model Exporter* component enables serialization of the current communication model and its integration into the platform-specific *Mashup Descriptor*. Apache Rave makes use of the OMDL[8] format to describe the configuration of UI mashups. However, the current specification doesn't provide any means to include IWC configuration into the mashup specification. Thus, we propose to extend OMDL documents with missing information on the *Communication Model*. An example of an extended *Mashup Descriptor* in OMDL format is given in Listing 1.2. The goal of the described mashup is to provide aggregated information on emergency incidents during natural disasters such as in case of flood. The mashup aggregates four widgets and defines IWC restrictions for the current composition. One of the widgets (flood graph) is completely isolated. Such configuration is useful to fix view of a widget and to avoid refreshes caused by changes others. Communication between the map widget and the contacts widget is forbidden only for the topic *http://example.org/phoneNumber*. It results in the behavior, that no marker selection in the map will refresh the contacts widget, while activities in other widgets may do.

Listing 1.2. OMDL mashup description and proposed extension

```
<workspace xmlns="http://omdl.org/"
    xmlns:oa="http://www.openajax.org/hub">
  <identifier>http://example.org/mashup/379</identifier>
  <title>Dresden Flood</title>
  <description></description>
  <creator>Alexey</creator>
  <date>2013-06-18T14:39:58+0200</date>
  <layout>THREE COLUMNS</layout>

  <app id="http://example.org/incidentMap-1">
    <link href="http://example.org/s/incidentsMap.wgt"
    type="application/widget" rel="source"/>
    <position>LEFT TOP</position>
  </app>

  <app id="http://example.org/contacts-1">
    <link href="http://example.org/s/contacts.wgt"
    type="application/widget" rel="source"/>
    <position>LEFT MIDDLE</position>
  </app>

  <app id="http://example.org/floodGraph-1">
    <link href="http://example.org/s/floodGraph.wgt"
    type="application/widget" rel="source"/>
```

[8] http://omdl.org

```
    <position>RIGHT TOP</position>
  </app>

  <app id=”http://example.org/floodGraph−2”>
    <link href=”http://example.org/s/floodGraph.wgt”
    type=”application/widget” rel=”source”/>
    <position>RIGHT MIDDLE</position>
  </app>

  <!−− Proposed extension specifying mashup IWC behaviour −−>

  <!−− Isolate a widget completely −−>
  <oa:pubsub−restriction
    oa:source=”*”
    oa:target=”http://example.org/floodGraph−1”
    oa:topicRef=”*”/>
  <oa:pubsub−restriction
    oa:source=”http://example.org/floodGraph−1”
    oa:target=”*”
    oa:topicRef=”*”/>

  <!−− Forbid two widgets to communicate over the specified topic −−>
  <oa:pubsub−restriction
    oa:source=”http://example.org/incidentMap−1”
    oa:target=”http://example.org/contacts−1”
    oa:topicRef=”http://example.org/phoneNumber”/>

</workspace>
```

This section presented an implementation of the reference architecture for the publish-subscribe choreography strategy. Application of the approach to other strategies requires to implement strategy-specific *Model Importer*, *Model Exporter* and *Message Broker* components. The *Awareness and Control Module* and *Communication Model*, however, can be reused across different platforms with minor adaptations.

4 Evaluation

To evaluate the presented approach in practice, we explored the following three hypotheses:

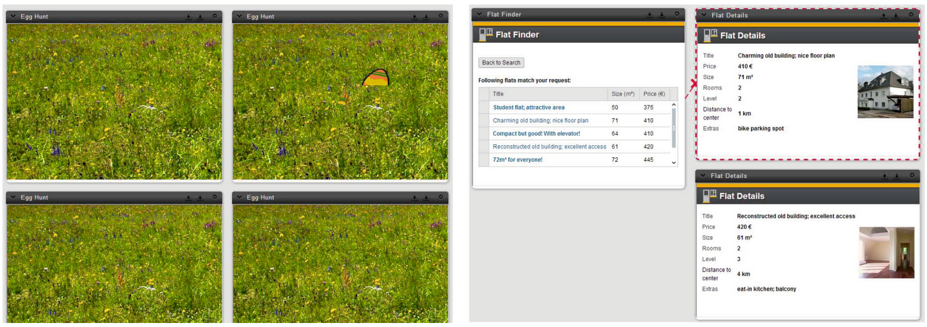
1. *Users solve tasks that require IWC faster if awareness and control mechanisms are used.* The intent was to check if the proposed mechanisms increase efficiency of end-users while working with UI mashups. The hypothesis was considered to be approved if the time needed to complete a given task with activated awareness and control facilities was lower than without them.

2. *Users find out easier, which widgets are connected, if IWC is visualized.* The intent was to check if the awareness mechanisms help users to spot, understand, and make use of the connections between widgets. The hypothesis was considered to be approved if average assessment on the ease of finding connections between widgets was higher with awareness facilities than without. The hypothesis was checked with two different types of widgets being employed: one group had mostly static content, while the other included animated changes. The rationale was that animated changes in widgets themselves might be sufficient on their own to identify communication partners, thereby making additional awareness mechanisms unnecessary.
3. *Users find the proposed control mechanisms easy to use.* The intent was to check if users felt comfortable with the proposed tailoring facilities. The approval condition was that more than 60% of participants agree or strongly agree that control facilities are easy to use.

To test the introduced hypotheses we applied the laboratory experiment methodology. Overall, 27 participants took part in the user study. Almost 90% of participants had no programming skills but were experts in the domain of marketing and telecommunication. The majority of users (74%) had an understanding of the term "widget", which was mostly related to mobile devices and the "Windows Vista Sidebar". Only 4 out of 27 users had ever configured a portal UI (mostly intranet portal) on their own, e. g., by repositioning of widgets and changing the colour scheme.

For each of the 27 participants, the evaluation procedure took about one hour and involved the following steps: Before the task execution, participants filled in a pre-evaluation questionnaire to judge their skill levels. Based on the results, they were evenly distributed over test and control groups. After that, users were given an introduction on the widget mashup platform, its purpose, concepts (mashups, widgets, etc.) and core functionalities. Following the introduction, users had the chance to explore and try out different aspects of the portal as they liked. After then, participants were asked to complete three tasks targeting different aspects of the system. The completion time was measured for each group and for each task. In a standardized post-questionnaire, users could express their subjective opinions on the introduced facilities. The study used a two-tailed t test, with a significance level of 95

Awareness Facilities. In the first experiment, users were asked to play a game on a dedicated mashup, called "Easter Egg Hunt" (cf. Figure 3, left). The goal of the game was to find the Easter egg in each of the nine visible widgets. Only one egg was visible at a time. Upon clicking it, the egg would disappear and appear in another widget. Thus, users quickly had to find the corresponding widget and the egg inside. For the test group, visualization of communicating widget pairs was enabled, so that outgoing and incoming data were displayed in the widget header. Thereby, the data flow could be perceived by users and, ideally, they could deduce where to look next. The control group accomplished the task without IWC visualization. The experiment was conducted in two different setups:



Flat Finder

Following flats match your request:

Title	Size (m²)	Price (€)
Student flat; attractive area	50	375
Charming old building; nice floor plan	71	410
Compact but good! With elevator!	64	410
Reconstructed old building; excellent access	61	420
72m² for everyone!	72	445

Flat Details

Title	Charming old building; nice floor plan
Price	410 €
Size	71 m²
Rooms	2
Level	2
Distance to center	1 km
Extras	bike parking spot

Flat Details

Title	Reconstructed old building; excellent access
Price	420 €
Size	61 m²
Rooms	2
Level	3
Distance to center	4 km
Extras	eat-in kitchen; balcony

Fig. 3. Evaluation mashups for testing end-user efficiency with awareness (left) and control (right) facilities

for animated content (the egg would "fall" into place) and static content (the egg would just appear). The time required for completion of the task was measured.

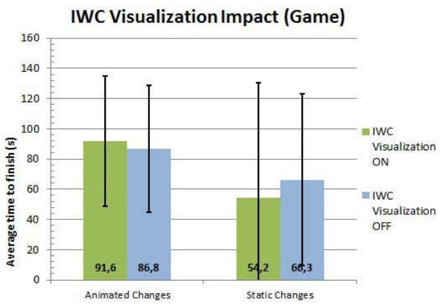
Control Facilities. In the second experiment, participants had to solve a comparison task using two widgets (cf. Figure 3, right). One widget gave an overview of flat offers in a selected city and the other one showed details of the offer. The task was to find the cheapest flat in one city and then a comparable one in another city. While the control group used the default setting with one "overview" and one "detail" widget, the test group was provided with two "detail" widgets, one of which they could isolate from communication to simplify the comparison. Once isolation was enabled for a detail widget, it would "freeze", so that new details could be loaded in the second widget and easily be compared to the first one. The time required for completion of the task was measured.

4.1 Results

Time measurements and evaluation of the post-questionnaire results yielded the following findings:

Hypotheses 1: End-User Efficiency. The results indicate a possible advantage of IWC visualization for widgets with mostly static content, i. e., whenever changes due to IWC are rather subtle as opposed to animations (cf. Figure 4, left). However, this difference is not statistically significant. When changes were animated by widgets, the average task completion times in test and control groups were roughly the same. Statistically, the test group was 18% slower than the control group (95% confidence), revealing a possible distraction of users due to the visualization.

This can be partly attributed to the overlapping of indicator flashes for fast users. The indicators were still flashing from their last interaction, when data was published by a new widget. This confused several users, who expected updated data, i. e., the egg, to appear in those widgets. The hypothesis for the awareness mechanisms is, thus, considered to be *not approved*.



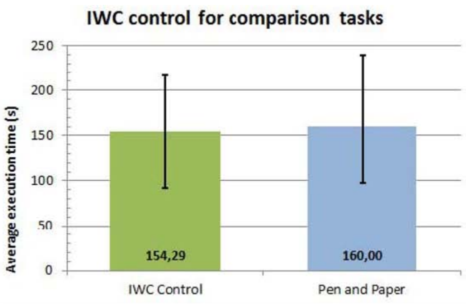
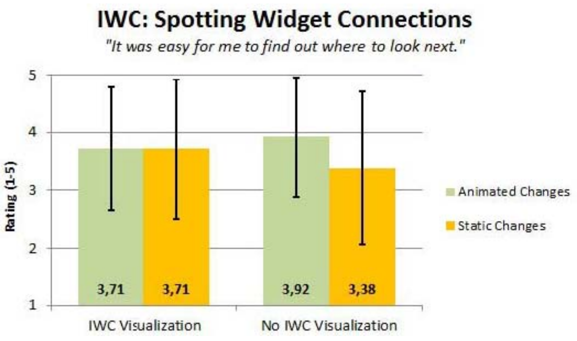


Fig. 4. Impact of awareness (left) and control (right) facilities on end-user efficiency

The control mechanisms for IWC, namely the possibility to isolate widgets, gave users a slight benefit when solving the flat comparison task (cf. Figure 4, right). As the time advantage is not significant, this hypothesis is also considered to be *not approved*.

Hypotheses 2: Usability of the Awareness Mechanisms. According to user ratings (cf. Figure 5, left), IWC visualization does not help users in subtly suggesting "where to look next" and thus understanding which widget are communicating. For users in the test group, it made no difference whether changes in widgets were animated or not. In contrast, users from the control group obviously found it easier to "follow the egg" if the changes were animated, since those were easier to spot in their peripheral field of vision. Thus, for mostly static widgets IWC visualization seems to compensate the missing IWC indicator and to facilitate the recognition of communicating parties.

In the light of the above results, the hypothesis is considered to be *approved* for widgets with mostly static content and *not approved* for widgets with animated changes. Based on this fact, we can derive a guideline for widget developers to animate changes triggered by IWC in order to improve usability of the future mashups.



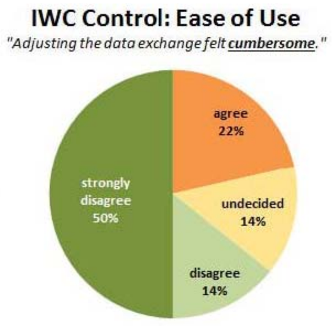


Fig. 5. Usability of the proposed awareness (left) and control (right) facilities

Hypotheses 3: Usability of the Control Mechanisms. IWC control mechanisms got very positive response. 64% of users found them easy to use (cf. Figure 5, right). The controls for this feature, namely the integration with the widget menu, were also rated positively. The hypothesis is considered to be *approved.*

In the post-questionnaire, the vast majority of the participants described the visualization and control facilities as helpful and easy to use. Users recommended making the IWC visualization optional and as such less subtle. One suggestion was to enable/disable visualization per widget by clicking the indicators directly. Furthermore, they suggested using a more noticeable colour scheme for indicating communication partners. It was proposed to investigate if a distinction between incoming/outgoing data is necessary to be visualized, or rather one indicator for taking part in data exchange is enough, e. g., a cogwheel. Finally, some participants suggested IWC controls, e. g., “Isolation”/”Pinning”, to be accessible from the widget header bar.

5 Related Work

The need for appropriate awareness and control facilities in choreographed user interface mashups has been recognized by a number of research projects and initiatives [8,21]. As for authors’ knowledge, the only hybrid mashup platform proposed at the moment is Chrooma+ [12], whereas only spare configuration of widget IWC behaviour is possible. In [21], Wilson proposes several ideas to tackle awareness and control challenges in UI mashups. Our work continues the research on this field and proposes actual solutions as well as corresponding architectural components.

The problem of visualizing and controlling interactions between autonomous entities is also tackled in other research areas. For example, in the field of self-organizing multi-agent systems, much research has been performed on the visualization of agent interactions. Typical strategies are, e.g., either to draw a graph of interactions in a whole system [1,6] or to highlight relationships between single agents and their environment.[14,17,15]. In [20], the authors propose to draw a causality graph to visualize message exchange in a multi-agent system. Though awareness of relationships in a multi-agent system can be increased with their approach, an a priori simulation and an event log are needed for the system to work. For mashups that employ cost-causing widgets (e.g., with telecommunication functionality), an a priori simulation might be undesirable.

An approach similar to the presented one is given in [16]. The authors propose a set of tools with different perspectives for monitoring and debugging of multi-agent systems. Relationships between agents and possible interactions are shown in the so-called ”society tool”. A requirement for the tool is that agents expose their partial knowledge about the outer world and communication capabilities to the tool, which then visualizes them in a graph fashion. Several controlling tools enable modification of agent states and configuration of their reactions on incoming messages. Though fine-grained exploration and adaptation of the system is possible, the target group of the approach are skilled developers.

In the field of natural programming environments, a common practice is to use natural language and question-answering games to explore a system. The WhyLine tool [11] applies natural language to enable unskilled developers to debug their algorithms. Using menus and pictograms of objects involved into an algorithm, developers can construct "why did" and "why did not" questions in order to explore system behaviour. A user study revealed that the participants were more efficient with this system than with traditional debugging tools. The approach is applied for explanation of static information and recorded event log. However, it doesn't foresee any means for visualization of active data transfer as it is required for IWC scenarios.

Similar research on control facilities in the context of loosely coupled communicating components can be found among visual programming tools. Lego-Mindstorms products[9] based on LabVIEW [10] enable unskilled developers to design their algorithms in a graphical way. Different boxes representing robot functionalities like move, rotate or stop can be connected with each other and controlled using loops or branches. The tool applies the "wiring" metaphor to connect inputs and outputs of the components, which makes the approach flexible and extensible. Many mashup platforms have adopted the "wiring" technique to enable their users to define data flow in compositions [9,23] and have shown its suitability in the context of end-user development [7]. However, the direct adoption of the technique to choreographed UI mashups is impossible due to the self-organizing nature of the aggregated widgets. The awareness and control facilities in this paper are inspired by the "wiring" approach and apply it to enable visualization and tailoring of the unified communication model.

6 Conclusions and Outlook

Missing understanding of inter-widget dependencies and lack of IWC control facilities can significantly impact usability and user experience within choreographed UI mashups. This paper presented an approach to systematically develop hybrid mashups with integrated IWC awareness and control mechanisms. The resulting solutions differ from the current state of the art in that they both enable self-emerging ("automatic" from end-users' point of view) IWC and keep users in control of how widgets communicate at the same time. The proposed reference architecture can be used as a guidance to build UI mashup platforms either from scratch or by extension of existing choreographed ones. One implementation of the reference architecture has been demonstrated in the context of an existing UI mashup platform based on publish-subscribe IWC strategy. The implementation is easily portable to other communication models such as RPC or shared memory.

A user study with 27 participants confirmed usability of the proposed control mechanisms and helped to discover shortcomings in the awareness ones.

[9] `http://education.lego.com/en-us/preschool-and-school/secondary/mindstorms-education-ev3`

[10] `http://www.ni.com/labview`

The drawn consequence for the future work is therefore to explore alternative non-ambiguous visualization techniques (e. g., flashing arrows instead of blinking icons) and to make the control mechanisms more prominent (e. g., by making the isolation icons accessible from the composition canvas). Finally, explanation of data being transferred between widgets has not been tackled sufficiently so far. The challenge here is to present the technical data (such as message syntax and semantic) in end-user-friendly way. Use of dedicated widget annotations or semantically enriched messages (as proposed in [18]) should be explored in the future.

Online Demonstration. A demonstration of the proposed awareness and control facilities integrated into Apache Rave and Apache Wookie projects is available at `http://vsr.cs.tu-chemnitz.de/demo/hybrid-ui-mashups`.

Acknowledgement. This work was supported by the European Commission (project OMELETTE, contract 257635). The authors thank Vadim Chepegin and TIE Kinetix b.v. for their contributions to the user study.

References

1. Bellifemine, F., Caire, G., Poggi, A., Rimassa, G.: JADE: A White Paper. EXP in Search of Innovation 3(3), 6–19 (2003)
2. Cappiello, C., Daniel, F., Matera, M., Picozzi, M., Weiss, M.: Enabling end user development through mashups: Requirements, abstractions and innovation toolkits. In: Piccinno, A. (ed.) IS-EUD 2011. LNCS, vol. 6654, pp. 9–24. Springer, Heidelberg (2011)
3. Cappiello, C., Matera, M., Picozzi, M., Sprega, G., Barbagallo, D., Francalanci, C.: DashMash: A mashup environment for end user development. In: Auer, S., Díaz, O., Papadopoulos, G.A. (eds.) ICWE 2011. LNCS, vol. 6757, pp. 152–166. Springer, Heidelberg (2011)
4. Chudnovskyy, O., Nestler, T., Gaedke, M., Daniel, F., Ignacio, J.: End-User-Oriented Telco Mashups: The OMELETTE Approach. In: WWW 2012 Companion Volume, pp. 235–238 (2012)
5. Chudnovskyy, O., Pietschmann, S., Niederhausen, M., Chepegin, V., Griffiths, D., Gaedke, M.: Awareness and control for inter-widget communication: Challenges and solutions. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 114–122. Springer, Heidelberg (2013)
6. Collis, J.C., Ndumu, D.T., Nwana, H.S., Lee, L.C.: The zeus agent building toolkit. BT Technology Journal 16(3), 60–68 (1998)
7. Imran, M., Soi, S., Kling, F., Daniel, F., Casati, F., Marchese, M.: On the systematic development of domain-specific mashup tools for end users. In: Brambilla, M., Tokuda, T., Tolksdorf, R. (eds.) ICWE 2012. LNCS, vol. 7387, pp. 291–298. Springer, Heidelberg (2012)
8. Isaksson, E., Palmer, M.: Usability and inter-widget communication in PLEs. In: Proceedings of the 3rd Workshop on Mashup Personal Learning Environments (2010)
9. JackBe. Presto Wires, `http://www.jackbe.com/products/wires.php`

10. Kirschenmann, U., Scheffel, M., Friedrich, M., Niemann, K., Wolpers, M.: Demands of modern pLEs and the ROLE approach. In: Wolpers, M., Kirschner, P.A., Scheffel, M., Lindstaedt, S., Dimitrova, V. (eds.) EC-TEL 2010. LNCS, vol. 6383, pp. 167–182. Springer, Heidelberg (2010)
11. Ko, A.J., Myers, B.A.: Designing the whyline: a debugging interface for asking questions about program behavior. In: Proceedings of the SIGCHI Conf. on Human Factors in Computing Systems, vol. 6, pp. 151–158 (2004)
12. Krug, M., Wiedemann, F., Gaedke, M.: Enhancing media enrichment by semantic extraction. In: Proceedings of the 23nd International Conference on World Wide Web Companion, WWW 2014 Companion (to appear, 2014)
13. Lizcano, D., Soriano, J., Reyes, M., Hierro, J.J.: Ezweb/fast: Reporting on a successful mashup-based solution for developing and deploying composite applications in the upcoming ubiquitous soa. In: Proceedings of the 2nd Intl. Conf. on Mobile Ubiquitous Computing Systems, Services and Technologies, pp. 488–495. IEEE (September 2008)
14. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: Mason: A multiagent simulation environment. Simulation 81(7), 517–527 (2005)
15. Minar, N., Burkhart, R., Langton, C.: The swarm simulation system: A toolkit for building multi-agent simulations. Technical report (1996)
16. Ndumu, D.T., Nwana, H.S., Lee, L.C., Collis, J.C.: Visualising and debugging distributed multi-agent systems. In: Proceedings of the Third Annual Conference on Autonomous Agents, AGENTS 1999, pp. 326–333. ACM, New York (1999)
17. North, M.J., Howe, T.R., Collier, N.T., Vos, J.R.: The Repast Simphony runtime system. In: Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms, ANL/DIS-06-1, co-sponsored by Argonne National Laboratory and The University of Chicago (2005)
18. Radeck, C., Blichmann, G., Meißner, K.: CapView – functionality-aware visual mashup development for non-programmers. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 140–155. Springer, Heidelberg (2013)
19. Rümpel, A., Radeck, C., Blichmann, G., Lorz, A., Meißner, K.: Towards do-it-yourself development of composite web applications. In: Proceedings of International Conference on Internet Technologies & Society 2011, pp. 330–332 (2011)
20. Vigueras, G., Botia, J.A.: Tracking causality by visualization of multi-agent interactions using causality graphs. In: Dastani, M., El Fallah Seghrouchni, A., Ricci, A., Winikoff, M. (eds.) ProMAS 2007. LNCS (LNAI), vol. 4908, pp. 190–204. Springer, Heidelberg (2008)
21. Wilson, S.: Design challenges for user-interface mashups user control and usability in inter-widget communications (2012)
22. Wilson, S., Daniel, F., Jugel, U., Soi, S.: Orchestrated user interface mashups using W3C widgets. In: Harth, A., Koch, N. (eds.) ICWE 2011. LNCS, vol. 7059, pp. 49–61. Springer, Heidelberg (2012)
23. Yahoo! Yahoo! Pipes, `http://pipes.yahoo.com/`
24. Zuzak, I., Ivankovic, M., Budiselic, I.: A classification framework for web browser cross-context communication. CoRR, abs/1108.4770 (2011)

Identifying Patterns in Eyetracking Scanpaths in Terms of Visual Elements of Web Pages

Sukru Eraslan[1,2], Yeliz Yesilada[1], and Simon Harper[2]

[1] Middle East Technical University, Northern Cyprus Campus, Guzelyurt, Mersin 10, Turkey
{seraslan,yyeliz}@metu.edu.tr

[2] University of Manchester, School of Computer Science, United Kingdom
sukru.eraslan@postgrad.manchester.ac.uk,
simon.harper@manchester.ac.uk

Abstract. Web pages are typically decorated with different kinds of visual elements that help sighted people complete their tasks. Unfortunately, this is not the case for people accessing web pages in constraint environments such as visually disabled or small screen device users. In our previous work, we show that tracking the eye movements of sighted users provide good understanding of how people use these visual elements. We also show that people's experience in constraint environments can be improved by reengineering web pages by using these visual elements. However, in order to reengineer web pages based on eyetracking, we first need to aggregate, analyse and understand how a group of people's eyetracking data can be combined to create a common scanpath (namely, eye movement sequence) in terms of visual elements. This paper presents an algorithm that aims to achieve this. This algorithm was developed iteratively and experimentally evaluated with an eyetracking study. This study shows that the proposed algorithm is able to identify patterns in eyetracking scanpaths and it is fairly scalable. This study also shows that this algorithm can be improved by considering different techniques for pre-processing the data, by addressing the drawbacks of using the hierarchical structure and by taking into account the underlying cognitive processes.

Keywords: eyetracking, scanpaths, commonality, transcoding, reengineering.

1 Introduction

Web pages mainly consist of different kinds of visual elements, such as menu, logo and hyperlinks. These visual elements help sighted people complete their tasks, but unfortunately small screen device users and disabled users cannot benefit from these elements. When people access web pages with small screen devices, they typically experience many difficulties [1]. For example, on small screen devices, only some parts of web pages are accessible or the complete web page is available with very small text size. Hence, they may need to scroll or zoom a lot which can be annoying. Moreover, they may need more time and effort to find their targets. Similarly, web experience can be challenging for visually disabled users who typically use screen readers to access the web [2]. Since screen readers follow the source code of web pages, visually disabled users have to listen to unnecessary clutter to get to the main content [3].

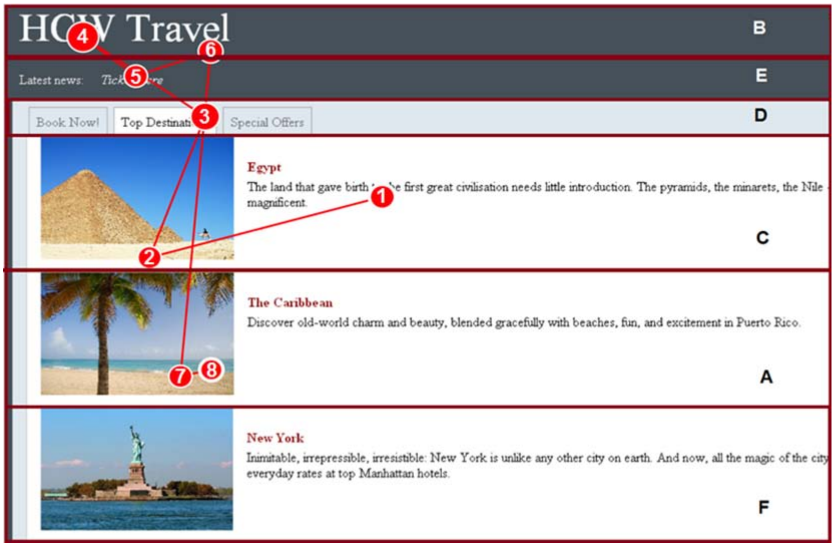

HCIW Travel
Latest news: Tick...
Book Now!
Top Destinati...
Special Offers
Egypt
The land that gave birth to the first great civilisation needs little introduction. The pyramids, the minarets, the Nile ... magnificent.
The Caribbean
Discover old-world charm and beauty, blended gracefully with beaches, fun, and excitement in Puerto Rico.
New York
Inimitable, irrepressible, irresistible: New York is unlike any other city on earth. And now, all the magic of the city ... everyday rates at top Manhattan hotels.
A
B
C
D
E
F
1
2
3
4
5
6
7
8

Fig. 1. A scanpath on a segmented web page

In our previous work, we show that reengineering web pages by using the visual elements can improve the user experience in constraint environments [4]. However, identifying visual elements and their role is the key for such reengineering process. To automatically process a web page and identify these elements, in our previous work we have extended and improved the Vision Based Page Segmentation (VIPS) algorithm [5,6]. This extended algorithm automatically discovers visual elements and relates them to the underlying source code. It allows direct access to these visual elements via XPath. However, this algorithm does not provide any information on how these visual elements are used. In our previous work, we also show that tracking the eye movements of sighted users provide good understanding of how they are used [2]. Eyes make quick movements which are called saccades. Between saccades, eyes make fixations where they become relatively stationary. Both fixations and saccades create scanpaths which are eye movement sequences [7]. Fig. 1 shows how a web page is segmented and illustrates a scanpath on a segmented web page. The circles represent fixations where the larger circles represent longer fixations. The numbers in the circles show the sequence. Also, the lines between circles are saccades.

In order to be able to use eyetracking data for reengineering web pages, this paper presents an algorithm called "*eMine scanpath algorithm*"[1]. This algorithm analyses and aggregates a group of people's eyetracking data to create a common scanpath in terms of visual elements of web pages (Section 3). Web pages are first automatically segmented into visual elements with the extended and improved version of the VIPS algorithm [6,5]. Eyetracking data is then exported and related to these visual elements. This creates individual scanpaths of users in terms of visual elements. These individual

[1] `http://emine.ncc.metu.edu.tr/`

scanpaths are then used by eMine scanpath algorithm to create a common scanpath. eMine scanpath algorithm was iteratively developed with the existing eyetracking data and our preliminary evaluation of this algorithm with the existing data was promising [8]. But in order to experientially evaluate validity and scalability of this algorithm, we conducted a new eyetracking study with 40 participants (Section 4). This study illustrates that eMine scanpath algorithm is able to identify a common scanpath in terms of visual elements of web pages and it is fairly scalable (Section 5 and Section 6). It has also revealed some weaknesses which can be improved in the future (Section 7).

2 Related Work

Eyetracking scanpaths have been analysed with different methods for different purposes. These methods typically use string representations of scanpaths which are generated using the sequence of Areas of Interest (AoIs) [9]. For example, the string representation of the scanpath in Fig. 1 is generated as CCDBEBAA. Different ways can be used to generate these AoIs such as using a grid layout directly [9] or the fixations' distribution over web pages [10]. However, these existing approaches typically treat a web page as an image to identify these AoIs which means these scanpaths cannot be used to process web pages. In order to address this, our previous work automatically segments a web page and each segment becomes an AoI [5,6]. This allows relating AoIs with the underlying source code which is important for being able to process web pages by using the eyetracking data.

The Levenshtein Distance (String-Edit) algorithm has commonly been used to analyse scanpaths [11,9]. This algorithm calculates the dissimilarity between the string representations of two scanpaths by transforming one to another with a minimum number of operations (insertion, deletion and substitution). For example, the dissimilarity between XYCZ and XYSZ is calculated as 1 (one) by the String-Edit algorithm because the substitution C with S is sufficient to transform one to another. Although the String-edit algorithm can be used to categorise scanpaths [12] and investigate differences between the behaviours of people on web pages [11], the algorithm itself is not able to identify a common scanpath for multiple scanpaths.

Transition Matrix is one of the methods which use multiple scanpaths to create a matrix [12]. This matrix allows identifying the possible next and previous AoI of the particular AoI. However, when this method is considered for identifying a common scanpath, some considerable problems arise, such as What is the start and end point of the common scanpath? Which probabilities should be considered?

To address these problems, some other methods can be considered. For example, the Shortest Common Supersequence method has been mentioned in literature to identify a common scanpath for multiple people but it has considerable weaknesses [13]. For example, it identifies XABCDEZ as a common scanpath for the individual scanpaths XAT, XBZ, XCZ, XDZ and XEZ. As can be easily recognised, the common scanpath is not supported by the individual scanpaths, for instance, the common scanpath has E which is included by only one individual scanpath (XEZ). Furthermore, the common scanpath is quite longer compared to the individual scanpaths.

Some methods, such as T-Pattern [14] and eyePatterns's Discover Patterns [12], have been proposed to detect subpatterns in eyetracking scanpaths. However, eyePatterns's Discover Patterns method [12] is not tolerant of extra items in scanpaths. For instance, XYZ can be detected as a subpattern for XYZ and WXYZ but it cannot be detected for XYZ and WXUYZ because of the extra item U. This shows that this method is reductionist which means it is likely to produce unacceptable short scanpaths.

The Multiple Sequence Alignment method was proposed to identify a common scanpath but this method was not validated [15]. Moreover, the Dotplots-based algorithm was proposed to identify a common scanpath for multiple people [16]. This algorithm creates a hierarchical structure by combining a pair of scanpaths with the Dotplots algorithm. The individual scanpaths are located at leafs whereas the common scanpath is located at the root. Some statistical methods have been applied to address the reductionist approach of the Dotplots algorithm [16].

We are interested in common patterns in eyetracking data instead of individual patterns to be able to reengineer web pages. However, as can be seen above, there is not much research in identifying common scanpaths and the existing ones are likely to produce unacceptable short common scanpaths. In this paper, we present our eMine scanpath algorithm to address the limitations of these existing approaches, especially the problem of being reductionist.

3 eMine Scanpath Algorithm

Algorithm 1 shows our proposed eMine scanpath algorithm [8] which takes a list of scanpaths and returns a scanpath which is common in all the given scanpaths. If there is only one scanpath, it returns that one as the common scanpath, if there is more than one scanpath, then it tries to find the most similar two scanpaths in the list by using the String-edit algorithm [11]. It then removes these two scanpaths from the list of scanpaths and introduces their common scanpath produced by the Longest Common Subsequence method [17] to the list of scanpaths. This continues until there is only one scanpath.

Algorithm 1. Find common scanpath

```
Input: Scanpath List
Output: Scanpath
 1: if the size of Scanpath List is equal to 1 then
 2:     return the scanpath in Scanpath List
 3: end if
 4: while the size of Scanpath List is not equal to 1 do
 5:     Find the two most similar scanpaths in Scanpath List with the String-edit algorithm
 6:     Find the common scanpath by using the Longest Common Subsequence method
 7:     Remove the similar scanpaths from the Scanpath List
 8:     Add the common scanpath to the Scanpath List
 9: end while
10: return the scanpath in Scanpath List
```

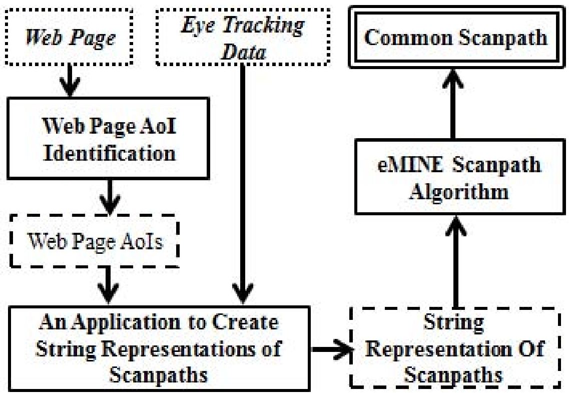

Web Page
Eye Tracking Data
Common Scanpath
Web Page AoI Identification
eMINE Scanpath Algorithm
Web Page AoIs
An Application to Create String Representations of Scanpaths
String Representation Of Scanpaths

Fig. 2. System architecture where '...' shows the input parts, '_ _ _' represents intermediate parts, '_' illustrates the functional parts and '=' is used for the output part

3.1 System Architecture and Implementation

eMine scanpath algorithm was integrated with the extended and improved version of the VIPS algorithm [6,5]. Fig. 2 illustrates the system architecture which consists of the following parts: two input parts (web page and eyetracking data), three functional parts (web page AoI identification, an application to create string representations of scanpaths, eMine scanpath algorithm), two intermediate parts which are created as an output of one functional part and used as an input for another functional part (web page AoIs, string representations of scanpaths) and one output part (common scanpath). The functional parts are explained below.

Web Page AoI Identification. A web page is used as an input for the web page AoI identification part. This part creates AoIs automatically by using the extended and improved version of the VIPS algorithm [6,5]. Even though, the extended VIPS was used, it would be easily replaced by an alternative method of AoI identification approach. These AoIs represent visual elements of web pages.

An Application to Create String Representations of Scanpaths. The automatically generated web page AoIs and eyetracking data, provided by eyetracking software, are then used by an application to create string representations of scanpaths.

eMine Scanpath Algorithm. Once the string representations are created, our scanpath algorithm is applied to them to produce a common scanpath in terms of AoIs.

eMine scanpath algorithm[2] was implemented on the Accessibility Tools Framework (ACTF)[3] which is an open-source Eclipse project.

[2] http://emine.ncc.metu.edu.tr/software.html
[3] http://www.eclipse.org/actf/

4 An Eyetracking Study

In order to experimentally evaluate validity and scalability of eMine scanpath algorithm, we conducted an eyetracking study. This study aims to investigate the following two research questions:

1. *Validity:* The aim is to investigate whether or not eMine scanpath algorithm can successfully identify common scanpaths in terms of visual elements of web pages. Thus, we ask *"Can eMine algorithm identify common scanpaths in terms of visual elements of web pages?"*.
2. *Scalability:* We would like to investigate whether or not eMine scanpath algorithm works well for different numbers of participants on different web pages. Hence, the research question here is *"How does the number of individual scanpaths affect common scanpaths?"*.

4.1 Equipment

Participants sat in front of a 17" monitor with a built-in TOBII T60 eye tracker with screen resolution 1280 x 1024. The web pages were on a HP ELiteBook 8530p laptop and these web pages were shown to the participants using the eye tracker's screen. Tobii Studio eye gaze analysis software was used to record the data. Eyetracking data was also stored on that laptop, too. The collected eyetracking data were analysed on a 17" monitor with the screen resolution 1280 x 1024.

4.2 Materials

Six web pages were randomly selected from a group of pages that were used in our previous study. That study focused on evaluating the extended and improved version of the VIPS algorithm and to have continuity in our studies we used same set of pages [6,5]. These web pages were categorised based on their complexity, which were low, medium and high [6,5,18]. Two web pages were chosen randomly from each level of complexity for our study. These pages with their complexity levels are as follow: Apple (Low), Babylon (Low), AVG (Medium), Yahoo (Medium), Godaddy (High) and BBC (High). Since the 5th segmentation granularity level was found as the most successful level with approximately 74% user satisfaction, we decided to use the 5th level for our experiments [6,5]. The segmented web pages can be seen in Fig. 3, 4, 5, 6, 7 and 8.

4.3 Procedure

This eyetracking study consists of the following three parts.

Introduction: The participants read the information sheet and signed the consent form. Next, they filled in the short questionnaire which was for the purpose of collecting basic demographic information of participants, which are gender, age groups and education level. The participants were also asked to rank their web page usage for the six web pages with 1 (Daily), 2 (Weekly), 3 (Monthly), 4 (Less than once a month) or 5 (Never).

Main Part: The participants sat in front of the eye tracker which calibrated to their gaze. They then viewed all of the six web pages twice, one view for searching (maximum 120 seconds) and one view for browsing in a random order. For browsing tasks, the participants were given 30 seconds as used in other studies [19]. The searching and browsing tasks are shown in Table 1. The researcher was responsible to check if the participants complete the tasks successfully and take notes if necessary.

Conclusion: At the end, the participants were asked to redraw three web pages from three different complexity levels.

4.4 User Tasks

User tasks are categorised into two groups for this study: searching and browsing. In the literature, many studies were conducted to categorise user tasks on the web [20]. G. Marchionini Search Activities Model is one of the most popular models in this field [20]. It consists of three groups which are lookup, learn and investigate [20]. Our searching category is related to fact finding which is associated with the lookup group whereas our browsing category is related to serendipitous browsing which is associated with the investigation group. The tasks which are defined for the six web pages are listed in Table 1.

We designed the system to ensure that half of the participants complete searching tasks firstly and then complete browsing tasks. Other half completed browsing task firstly and then completed searching tasks. The reason is to prevent familiarity effects on eye movements which can be caused by the user tasks.

4.5 Participants

The majority of the participants comprised students, along with some academic and administrative staff at Middle East Technical University Northern Cyprus Campus and the University of Manchester. Twenty male and twenty female volunteers participated. One male participant changed his body position during the study, so the eye tracker could not record his eye movements. Another male participant had no successful eye calibration. Unfortunately, these two participants were excluded from the study. Therefore, the eyetracking data of 18 males and 20 females were used to evaluate eMine scanpath algorithm.

All of the participants use the web daily. Most of the participants (18 participants) are aged between 18 and 24 years old, then 25-34 group (14 participants) and 35-54 group (6 participants). Moreover, 14 participants completed their high/secondary schools, 6 participants have a bachelor's degree, 9 participants have a master's degree and 9 participants completed their doctorate degrees.

5 Results

In this section, we present the major findings of this study in terms of the two research questions presented in Section 4.

Table 1. Tasks used in the eyetracking study

Apple	
Browsing	1. Can you scan the web page if you find something interesting for you?
Searching	1. Can you locate a link which allows watching the TV ads relating to iPad mini? 2. Can you locate a link labelled iPad on the main menu?
Babylon	
Browsing	1. Can you scan the web page if you find something interesting for you?
Searching	1. Can you locate a link you can download the free version of Babylon? 2. Can you find and read the names of other products of Babylon?
Yahoo	
Browsing	1. Can you scan the web page if you find something interesting for you?
Searching	1. Can you read the titles of the main headlines which have smaller images? 2. Can you read the first item under News title?
AVG	
Browsing	1. Can you scan the web page if you find something interesting for you?
Searching	1. Can you locate a link which you can download a free trial of AVG Internet Security 2013? 2. Can you locate a link which allows you to download AVG Antivirus FREE 2013?
GoDaddy	
Browsing	1. Can you scan the web page if you find something interesting for you?
Searching	1. Can you find a telephone number for technical support and read it? 2. Can you locate a text box where you can search a new domain?
BBC	
Browsing	1. Can you scan the web page if you find something interesting for you?
Searching	1. Can you read the first item of Sport News? 2. Can you locate the table that shows market data under Business title?

5.1 Validity

"Can eMine scanpath algorithm identify common scanpaths in terms of visual elements of web pages?"

The participants were asked to complete some searching tasks on web pages, therefore we are expecting to see that the common scanpath supports those tasks. We used eMine scanpath algorithm to identify a common scanpath for each of the six web pages. Some participants could not complete the searching tasks successfully and/or had calibration problems. These participants were defined as unsuccessful participants and excluded from the study. The success rates in completing searching tasks are as follow: Apple: 81.58 %, Babylon: 94.74 %, AVG: 94.74 %, Yahoo: 84.21 %, Godaddy: 73.68 % and BBC: 100 %. These values are calculated by dividing the number of the successful participants by the total number of the participants on the page.

Table 2 shows the common scanpaths and the abstracted common scanpaths produced by eMine scanpath algorithm for the web pages where 'P' represents the number of successful participants. In order to have abstracted common scanpaths, their string representations are simplified by abstracting consecutive repetitions [21,22]. For instance, MMPPQRSS becomes MPQRS.

Table 2. The common scanpaths produced by eMine scanpath algorithm for the web pages

Page Name	P	Common Scanpath	Abstracted Common Scanpath
Apple	31	EEB	EB
Babylon	36	MMPPQRSS	MPQRS
AVG	36	GGGGGGGGGGGGGGIIIIIIIII	GI
Yahoo	32	IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII	I
Godaddy	28	OOOOMMMMMM	OM
BBC	38	RNNNNN	RN

On the Apple web page, 31 out of 38 participants were successful. On this page, the participants were asked to locate a link which allows watching the TV ads relating to iPad mini and then locate a main menu item iPad. EB is identified as a common scanpath for these participants. Since E is associated with the first part and B is related to the second part of the searching task, this common scanpath completely supports the searching task. Fig. 3 shows this common scanpath on the Apple web page.

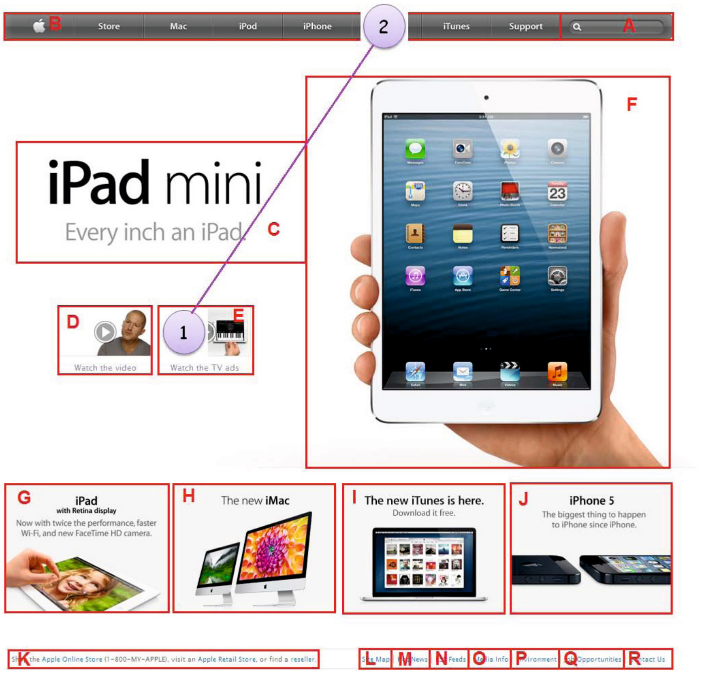
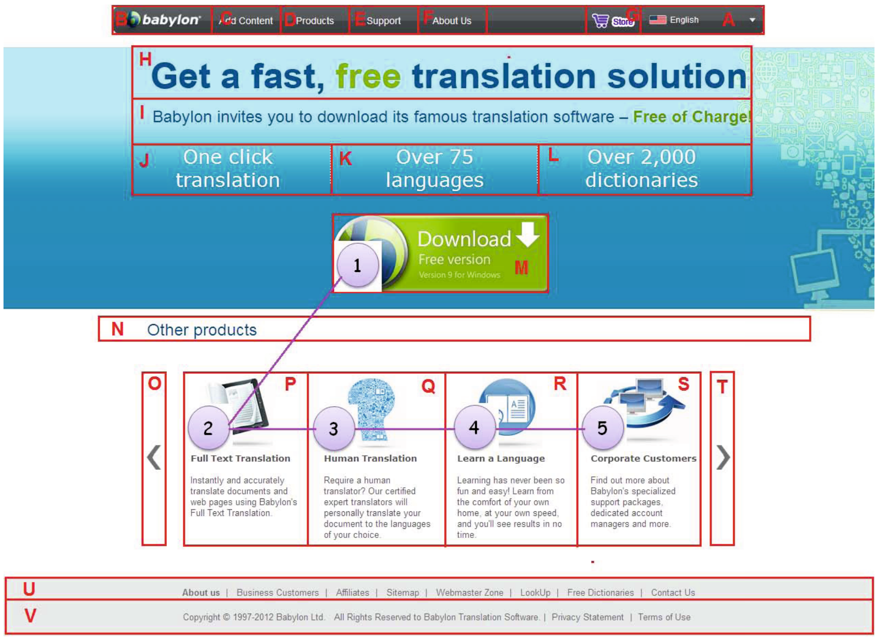


Fig. 3. Common scanpath on the Apple web page

On the Babylon web page, only 2 participants out of 38 were not successful. On this page, the participants were requested to locate a link which allows downloading a free version of Babylon and then read the names of other products of Babylon. The common scanpath for the 36 participants was identified as MPQRS shown in Fig. 4. M is related with a free version of Babylon whereas P, Q, R and S are associated with four other products of Babylon. Therefore, the common scanpath thoroughly supports the searching task.



Get a fast, free translation solution

Babylon invites you to download its famous translation software – Free of Charge!

Instantly and accurately translate documents and web pages using Babylon's Full Text Translation.

Require a human translator? Our certified expert translators will personally translate your document to the languages of your choice.

Learning has never been so fun and easy! Learn from the comfort of your own home, at your own speed, and you'll see results in no time.

Find out more about Babylon's specialized support packages, dedicated account managers and more.

Copyright © 1997-2012 Babylon Ltd. All Rights Reserved to Babylon Translation Software.

Fig. 4. Common scanpath on the Babylon web page

Similar to the Babylon web page, only 2 participants were unsuccessful on the AVG web page. The searching task here was locating a link which allows downloading a free trial of AVG Internet Security 2013 and then locating a link which allows downloading AVG Antivirus FREE 2013. The common scanpath was produced as GI where G has a link to download a free trial of AVG Internet Security 2013 and I contains a link to download AVG Antivirus FREE 2013. Therefore, the common scanpath, shown in Fig. 5, entirely supports the searching task.

For the Yahoo web page, 6 participants could not be successful. The participants required to read the titles of the main headlines which have smaller images and then read the first item under News title. Since only I is produced as a common scanpath on this web page and I contains both parts of the task, the common scanpath nicely supports the searching task, too. Fig. 6 shows this common scanpath.

Since 28 out of 38 participants were successful, 10 participants were excluded for the Godaddy web page. The successful participants read the telephone number for technical support and then located a text box where they can search for a new domain. eMine scanpath algorithm produced OM as a common scanpath shown in Fig. 7. Since M

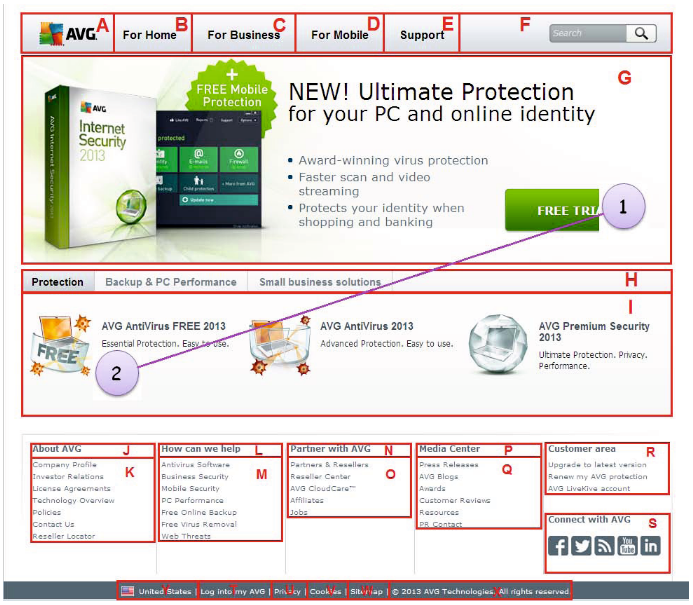


Fig. 5. Common scanpath on the AVG web page

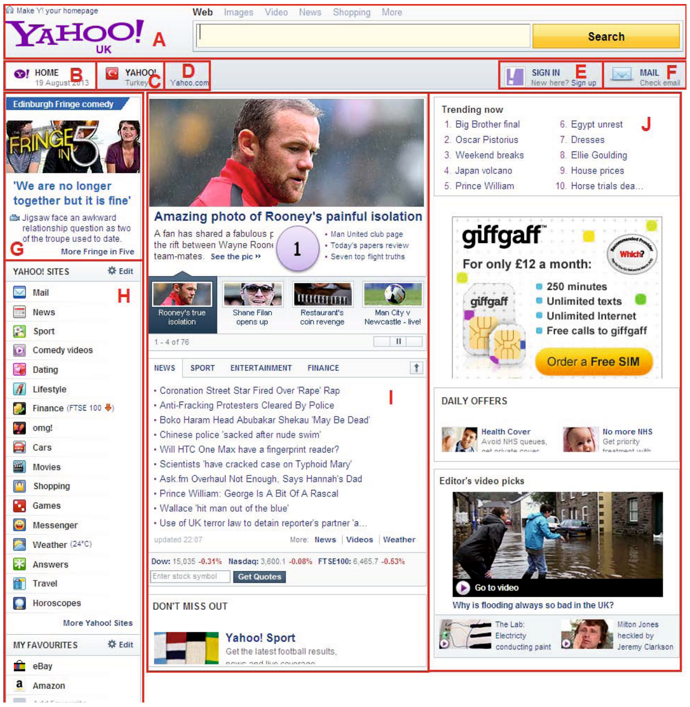

Make Y! your homepage
YAHOO! UK A
Web Images Video News Shopping More
Search
HOME 19 August 2013 B
YAHOO! Turkey C
D Yahoo.com
SIGN IN New here? Sign up E
MAIL Check email F
Edinburgh Fringe comedy
'We are no longer together but it is fine'
Jigsaw face an awkward relationship question as two of the troupe used to date.
More Fringe in Five
G
YAHOO! SITES Edit
H
Mail
News
Sport
Comedy videos
Dating
Lifestyle
Finance (FTSE 100)
omg!
Cars
Movies
Shopping
Games
Messenger
Weather (24°C)
Answers
Travel
Horoscopes
More Yahoo! Sites
MY FAVOURITES Edit
eBay
Amazon
Amazing photo of Rooney's painful isolation
A fan has shared a fabulous p... the rift between Wayne Roone... team-mates. See the pic »
1
• Man United club page
• Today's papers review
• Seven top flight truths
Rooney's true isolation
Shane Filan opens up
Restaurant's coin revenge
Man City v Newcastle - live!
1 - 4 of 76
NEWS SPORT ENTERTAINMENT FINANCE
I
• Coronation Street Star Fired Over 'Rape' Rap
• Anti-Fracking Protesters Cleared By Police
• Boko Haram Head Abubakar Shekau 'May Be Dead'
• Chinese police 'sacked after nude swim'
• Will HTC One Max have a fingerprint reader?
• Scientists 'have cracked case on Typhoid Mary'
• Ask.fm Overhaul Not Enough, Says Hannah's Dad
• Prince William: George Is A Bit Of A Rascal
• Wallace 'hit man out of the blue'
• Use of UK terror law to detain reporter's partner 'a...
updated 22:07 More: News | Videos | Weather
Dow: 15,035 -0.31% Nasdaq: 3,600.1 -0.08% FTSE100: 6,465.7 -0.53%
Enter stock symbol Get Quotes
DON'T MISS OUT
Yahoo! Sport
Get the latest football results, news and live coverage
Trending now J
1. Big Brother final
2. Oscar Pistorius
3. Weekend breaks
4. Japan volcano
5. Prince William
6. Egypt unrest
7. Dresses
8. Ellie Goulding
9. House prices
10. Horse trials dea...
giffgaff
For only £12 a month:
250 minutes
Unlimited texts
Unlimited Internet
Free calls to giffgaff
Order a Free SIM
DAILY OFFERS
Health Cover Avoid NHS queues, get private cover
No more NHS Get priority treatment with
Editor's video picks
Go to video
Why is flooding always so bad in the UK?
The Lab: Electricity conducting paint
Milton Jones heckled by Jeremy Clarkson

Fig. 6. Common scanpath on the Yahoo web page

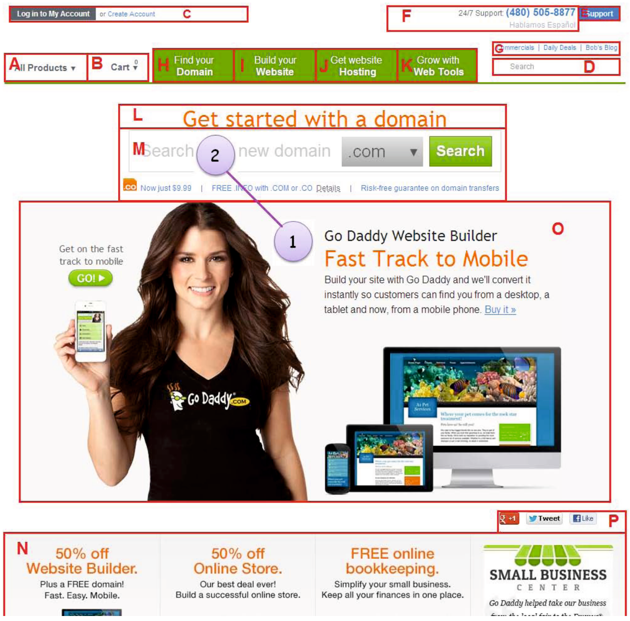


Fig. 7. Common scanpath on the Godaddy web page

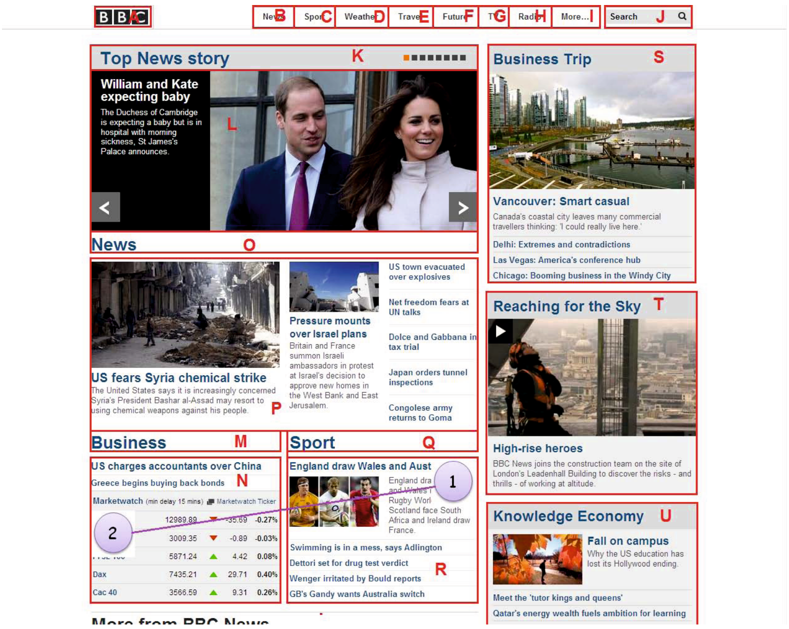


Fig. 8. Common scanpath on the BBC web page

contains the text box and there is no AoI in the scanpath which is related with the telephone number, the common scanpath partially supports the searching task on the Godaddy web page.

On the BBC web page, all participants completed the searching task successfully. The participants were asked to read the first item of the sports news and then locate a table which shows the market data. Therefore, the participants needed to locate R and then N. As the common scanpath RN is produced, it supports the searching task very well. Fig. 8 illustrates this common scanpath on the BBC web page.

To sum up, the common scanpaths on the Apple, Babylon, AVG, Yahoo and BBC web pages completely support the searching tasks whereas the common scanpath on the Godaddy web page partially supports the searching task.

5.2 Scalability

"How does the number of individual scanpaths affect common scanpaths?"

In order to test whether or not eMine scanpath algorithm works well with different numbers of individual scanpaths, we tested the algorithm with different numbers of individual participants. The participants were selected randomly from all of the successful participants. Table 3 illustrates the common scanpaths in terms of AoIs on the different web pages for 10, 20, 30 and 30+ participants while browsing and searching.

Table 3. The common scanpaths on the different web pages for 10, 20, 30 and 30+ participants while browsing and searching where '-' means that there was no sufficient number of successful participants and '——' means that no common scanpath was detected

Task	Page Name	P=10	P=20	P=30	P=30+
Browsing	Apple	IF	F	F	F
	Babylon	MS	M	M	M
	AVG	GIG	G	G	G
	Yahoo	IJI	I	I	I
	Godaddy	O	O	O	O
	BBC	LP	LP	P	——
Searching	Apple	EB	EB	EB	EB
	Babylon	MPQRS	MPQRS	MPQRS	MPQRS
	AVG	IGI	GI	GI	GI
	Yahoo	I	I	I	I
	Godaddy	OM	OM	-	-
	BBC	LPRN	RN	RN	RN

In order to see how the common scanpaths are affected when the number of participants increases, we calculated the similarities between the scanpaths which were produced for 10, 20, 30 and 30+ participants. To calculate the similarity between two common scanpaths the String-edit distance between two common scanpaths is divided by the length of the longer common scanpath to have a normalised score [23]. The purpose of a normalised score is to prevent any inconsistencies in similarities caused by different lengths [23,24]. Finally, the normalised score is subtracted from 1 [23]. For example, the common scanpath for 10 participants is LPRN and the common scanpath for 20 participants is RN on the BBC web page for the searching task. The String-edit distance is calculated as 2 between two scanpaths. After that, since the length of the longer scanpath (LPRN) is equal to 4, this distance is divided by 4. As a result, the normalised score is equal to 0.5. To calculate the similarity 0.5 is subtracted from 1, so the similarity between the two common scanpaths is equal to 0.5 (50 %). Table 4 shows these similarities between the common scanpaths for the searching task on the BBC web page whereas Table 5 illustrates the similarities between the common scanpaths for the browsing task on the Yahoo web page as examples.

Table 4. The similarities between the common scanpaths on the BBC web page for 10, 20, 30 and 30+ participants while searching

BBC Searching	P = 10	P = 20	P = 30	P = 30+
P = 10	—	50	50	50
P = 20	50	—	100	100
P = 30	50	100	—	100
P = 30+	50	100	100	—

Table 5. The similarities between the common scanpaths on the Yahoo web page for 10, 20, 30 and 30+ participants while browsing

Yahoo Searching	P = 10	P = 20	P = 30	P = 30+
P = 10	—	33.3	33.3	33.3
P = 20	33.3	—	100	100
P = 30	33.3	100	—	100
P = 30+	33.3	100	100	—

For both the browsing and searching tasks, we calculated the average similarity between the common scanpaths on each web page. To calculate these average similarities we divided the sum of the similarities between the scanpaths for 10, 20, 30 and 30+ participants by the total number of the similarities. In addition, we calculated the average similarity for both the browsing and searching tasks. Since each web page typically has four scanpaths (for 10, 20, 30 and 30+ participants), we determined their weights based on the number of scanpaths. All of the pages' weights are set to 4, except the Godaddy page because of the searching task. The Godaddy page has one common scanpath for 10 participants and one common scanpath for 20 participants, therefore its weight is set to 2. When the average is calculated, we multiplied the value with its weight to find the weighted value. After that, we found the sum of the weighted value and divided it by the sum of the weights. It was found that the average similarity for searching tasks (92.42%) is higher than the average similarity for the browsing task (69.44 %).

Table 6. The average of the similarities between the common scanpaths on each web page for 10, 20, 30 and 30+ participants

Page Name	Task	Average Similarity for Each Page
Apple	Browsing	75
Babylon	Browsing	75
AVG	Browsing	66.65
Yahoo	Browsing	66.65
Godaddy	Browsing	100
BBC	Browsing	33.33
Average Similarity for the 6 Pages	Browsing	69.44
Apple	Searching	100
Babylon	Searching	100
AVG	Searching	83.3
Yahoo	Searching	100
Godaddy	Searching	100
BBC	Searching	75
Average Similarity for the 6 Pages	Searching	92.42

6 Discussion

The eMine scanpath algorithm was experimentally evaluated with an eyetracking study and this study illustrates that the algorithm is able to successfully identify common scanpaths in terms of visual elements of web pages and it is fairly scalable.

The searching tasks completed by the participants on the given pages were used to validate eMine scanpath algorithm. We expected that the common scanpaths should support these searching tasks. For instance, on the Babylon web page, the participants were asked to locate the link which allows downloading the free version of Babylon (related to AoI M) and then read the names of other products of Babylon (related to AoIs P, Q, R and S). Therefore, we expected that the common scanpath on the Babylon web page should involve at least MPQRS for the searching tasks.

The results in Section 5.1 show that the common scanpaths produced by eMine scanpath algorithm completely support these tasks, except the common scanpath on the Godaddy page. On that page, the participants were asked to read a telephone number for technical support and locate the text box where they can search for a new domain. The common scanpath involves the AoI for the text box but does not include the AoI for the telephone number. Thus, it partially supports the searching task. There may be various reasons: (1) The participants might make a very few fixations on that AoI (2) Some participants might find the telephone number directly whereas some of them looked at many AoIs to find the telephone number. Therefore, it would be good to pre-process eyetracking data in depth to investigate the individual differences and their reasons.

Some other methods could also be used to validate eMine scanpath algorithm. One might consider calculating the similarities between the individual scanpaths and the common scanpath. Besides, the AoIs appeared in all individual scanpaths might be detected and then one part of the validation process could be done by using these AoIs.

The scalability of eMine scanpath algorithm was tested by using the different numbers of individual scanpaths as mentioned in Section 5.2. As expected, we can see that the algorithm is more scalable with the searching tasks because the participants were asked to complete some specific searching tasks. The average similarity is equal to 92.42 % between the common scanpaths which were produced with the different number of scanpaths for the searching tasks. However, the average similarity is equal to 69.44 % for the browsing tasks. Based on these values we can suggest that our algorithm is fairly scalable, especially in searching tasks.

There are some differences between scanpaths, such as producing LPRN for 10 participants and RN for 30+ participants on the BBC page. It is caused by using the hierarchical structure. As mentioned in Section 3, eMine scanpath algorithm uses a hierarchical structure while identifying common scanpaths. It selects the two most similar scanpaths from the list and finds their longest common subsequence. It is iteratively repeated until a single scanpath left. Because of the hierarchical structure, some information in intermediate levels can be lost because of combining two scanpaths.

Assume that there are three sequences: S1: GATACCAT S2: CTAAAGTC and S3: GCTATTGCG [17]. S1 and S2 can be aligned firstly and then S1'= - - A - A - - A - - - can obtained [17]. Following this, S1' and S3 can be aligned and then S3'= - - - A - - - - - - - - can be obtained [17]. This example clearly illustrates that the hierarchical structure can make the method reductionist. Here, all of the three scanpaths have G and T in different locations but G and T do not exist at the end. This may cause some differences in common scanpaths. Because of this reason, eMine scanpath algorithm was not able to identify any common scanpath on the BBC page for the browsing task. When a number of individual scanpaths is increased, the different most similar scanpath pairs can be generated and this may affect common scanpaths. Although eMINE scanpath algorithm has some drawbacks because of the hierarchical structure, it still partly addresses the reductionist problem of the other existing approaches (See Section 2).

To address the drawbacks of using the hierarchical structure a constraint might be created to prevent losing the AoIs appeared in all individual scanpaths in intermediate levels. Alternatively, some statistical approaches can be used to sort these AoIs and then create a common scanpath for multiple people.

7 Concluding Remarks and Future Work

This paper presents an algorithm and its evaluation that identifies common scanpaths in terms of visual elements of web pages. These visual elements are first automatically generated with the extended and improved version of the VIPS algorithm [6,5]. Eyetracking data is then related to these visual elements and individual scanpaths are created in terms of these visual elements. This algorithm then uses these individual scanpaths and generates a common scanpath in terms of these visual elements. This common scanpath can be used for reengineering web pages to improve the user experience in constraint environments.

To our knowledge, there is no work on correlating scanpaths with visual elements of web pages and the underlying source code, and this work is novel from that perspective [6,5]. This paper also shows how the validity and scalability of eMine scanpath algorithm was demonstrated with an eyetracking study. The results clearly show that this algorithm is able to identify common scanpaths in terms of visual elements of web pages and it is fairly stable. This algorithm aims to address the reductionist problem that the other existing work has, but the results show that there is still room for improvement.

The eyetracking study also suggests some directions for future work. It indicates that the individual differences can affect the identification of patterns in eyetracking scanpaths. Thus, eyetracking data should be pre-processed to investigate the individual differences and their reasons. Since an eye tracker collects a large amount of data, pre-processing is also required to eliminate noisy data. It is important because noisy data are likely to decrease the commonality in scanpaths. Another benefit of pre-processing is to identify outliers which are potential to decrease the commonality, too.

Finally, as with the existing scanpath methods, eMine scanpath algorithm also tends to ignore the complexities of the underlying cognitive processes. However, when people follow a path to complete their tasks on web pages, there may be some reasons that affect their decisions. Underlying cognitive processes can be taken into account while identifying common scanpaths.

Acknowledgments. The project is supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) with the grant number 109E251. As such the authors would like to thank to TÜBİTAK for their continued support. We would also like to thank our participants for their time and effort.

References

1. W3C WAI Research and Development Working Group (RDWG): Research Report on Mobile Web Accessibility. In: Harper, S., Thiessen, P., Yesilada, Y., (eds.): W3C WAI Symposium on Mobile Web Accessibility. W3C WAI Research and Development Working Group (RDWG) Notes. First public working draft edn. W3C Web Accessibility Initiative (WAI) (December 2012)
2. Yesilada, Y., Jay, C., Stevens, R., Harper, S.: Validating the Use and Role of Visual Elements of Web Pages in Navigation with an Eye-tracking Study. In: The 17th international Conference on World Wide Web, WWW 2008, pp. 11–20. ACM, New York (2008)
3. Yesilada, Y., Stevens, R., Harper, S., Goble, C.: Evaluating DANTE: Semantic Transcoding for Visually Disabled Users. ACM Trans. Comput.-Hum. Interact. 14(3), 14 (2007)
4. Brown, A., Jay, C., Harper, S.: Audio access to calendars. In: W4A 2010, pp. 1–10. ACM, New York (2010)
5. Akpınar, M.E., Yesilada, Y.: Heuristic Role Detection of Visual Elements of Web Pages. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 123–131. Springer, Heidelberg (2013)
6. Akpınar, M.E., Yesilada, Y.: Vision Based Page Segmentation Algorithm: Extended and Perceived Success. In: Sheng, Q.Z., Kjeldskov, J. (eds.) ICWE Workshops 2013. LNCS, vol. 8295, pp. 238–252. Springer, Heidelberg (2013)
7. Poole, A., Ball, L.J.: Eye tracking in human-computer interaction and usability research: Current status and future Prospects. In: Ghaoui, C. (ed.) Encyclopedia of Human-Computer Interaction. Idea Group, Inc., Pennsylvania (2005)
8. Yesilada, Y., Harper, S., Eraslan, S.: Experiential transcoding: An EyeTracking approach. In: W4A 2013, p. 30. ACM (2013)

9. Takeuchi, H., Habuchi, Y.: A quantitative method for analyzing scan path data obtained by eye tracker. In: CIDM 2007, April 1-5, pp. 283–286 (2007)
10. Santella, A., DeCarlo, D.: Robust clustering of eye movement recordings for quantification of visual interest. In: ETRA 2004, pp. 27–34. ACM, New York (2004)
11. Josephson, S., Holmes, M.E.: Visual attention to repeated internet images: testing the scanpath theory on the world wide web. In: ETRA 2002, pp. 43–49. ACM, NY (2002)
12. West, J.M., Haake, A.R., Rozanski, E.P., Karn, K.S.: EyePatterns: software for identifying patterns and similarities across fixation sequences. In: ETRA 2006, pp. 149–154. ACM, New York (2006)
13. Räihä, K.-J.: Some applications of string algorithms in human-computer interaction. In: Elomaa, T., Mannila, H., Orponen, P. (eds.) Ukkonen Festschrift 2010. LNCS, vol. 6060, pp. 196–209. Springer, Heidelberg (2010)
14. Mast, M., Burmeister, M.: Exposing repetitive scanning in eye movement sequences with t-pattern detection. In: IADIS IHCI 2011, Rome, Italy, pp. 137–145 (2011)
15. Hembrooke, H., Feusner, M., Gay, G.: Averaging scan patterns and what they can tell us. In: ETRA 2006, p. 41. ACM, New York (2006)
16. Goldberg, J.H., Helfman, J.I.: Scanpath clustering and aggregation. In: ETRA 2010, pp. 227–234. ACM, New York (2010)
17. Chiang, C.H.: A Genetic Algorithm for the Longest Common Subsequence of Multiple Sequences. Master's thesis, National Sun Yat-sen University (2009)
18. Michailidou, E.: ViCRAM: Visual Complexity Rankings and Accessibility Metrics. PhD thesis, University of Manchester (2010)
19. Jay, C., Brown, A.: User Review Document: Results of Initial Sighted and Visually Disabled User Investigations. Technical report, University of Manchester (2008)
20. Marchionini, G.: Exploratory Search: From Finding to Understanding. Commun. ACM 49(4), 41–46 (2006)
21. Brandt, S.A., Stark, L.W.: Spontaneous Eye Movements During Visual Imagery Reflect the Content of the Visual Scene. J. Cognitive Neuroscience 9(1), 27–38 (1997)
22. Jarodzka, H., Holmqvist, K., Nyström, M.: A Vector-based, Multidimensional Scanpath Similarity Measure. In: ETRA 2010, pp. 211–218. ACM, New York (2010)
23. Foulsham, T., Underwood, G.: What can Saliency Models Predict about Eye Movements? Spatial and Sequential Aspects of Fixations during Encoding and Recognition. Journal of Vision 8(2), 1–17 (2008)
24. Cristino, F., Mathot, S., Theeuwes, J., Gilchrist, I.D.: Scanmatch: a novel method for comparing fixation sequences. Behavior Research Methods 42(3), 692–700 (2010)

Identifying Root Causes of Web Performance Degradation Using Changepoint Analysis

Jürgen Cito[1], Dritan Suljoti[2], Philipp Leitner[1], and Schahram Dustdar[3]

[1] s.e.a.l. – Software Evolution & Architecture Lab, University of Zurich, Switzerland
{cito,leitner}@ifi.uzh.ch
[2] Catchpoint Systems, Inc., New York, USA
drit@catchpoint.com
[3] Distributed Systems Group, Vienna University of Technology, Austria
dustdar@dsg.tuwien.ac.at

Abstract. The large scale of the Internet has offered unique economic opportunities, that in turn introduce overwhelming challenges for development and operations to provide reliable and fast services in order to meet the high demands on the performance of online services. In this paper, we investigate how performance engineers can identify three different classes of externally-visible performance problems (global delays, partial delays, periodic delays) from concrete traces. We develop a simulation model based on a taxonomy of root causes in server performance degradation. Within an experimental setup, we obtain results through synthetic monitoring of a target Web service, and observe changes in Web performance over time through exploratory visual analysis and changepoint detection. Finally, we interpret our findings and discuss various challenges and pitfalls.

1 Introduction

The large scale of the Internet has offered unique economic opportunities by enabling the ability to reach a tremendous, global user base for businesses and individuals alike. The great success and opportunities also open up overwhelming challenges due to the drastic growth and increasing complexity of the Internet in the last decade. The main challenge for development and operations is to provide reliable and fast service, despite of fast growth in both traffic and frequency of requests. When it comes to speed, Internet users have high demands on the performance of online services. Research has shown that nowadays 47% of online consumers expect load times of *two seconds or less* [7, 14, 21]. With the growth of the Internet and its user base, the underlying infrastructure has drastically transformed from single server systems to heterogeneous, distributed systems. Thus, the end performance depends on diverse factors in different levels of server systems, networks and infrastructure, which makes providing a satisfying end-user experience and QoS (Quality of Service) a challenge for large scale Internet applications. Generally, providing a consistent QoS requires continually collecting data on Web performance on the Web service provider side, in order to observe and track changes in desired metrics, e.g., service response time. Reasons to observe these changes are different in their nature, and range from detecting anomalies, identifying patterns, ensuring service reliability, measuring performance changes after new software releases, or discovering performance degradation.

Early detection and resolution of root causes of performance degradations can be achieved through monitoring of various components of the system. Monitoring can be classified as either *active* or *passive* monitoring, and, orthogonally, as *external* or *internal.* In active monitoring, monitoring agents are actively trying to connect to the target system in order to collect performance data, whether the system is accessed by real end-users or not. Passive monitoring, on the other hand, only collects measurements if the system is actively used. Internal and external monitoring differentiate in whether the measurements are obtained in systems within the organization's data center or through end-to-end monitoring over the network outside the data center. This has ramifications in terms of what the level of detail of monitoring data that is available. In our experiments, we make use of active, external monitoring, which provides a way of capturing an end user perspective and enables the detection of issues before they affect real users [17].

Whatever the reason to observe changes may be, the measurements are only useful when we know how to properly analyze them and turn our data into informed decisions. The main contribution of this paper is a model for understanding performance data via analyzing how common underlying root causes of Web performance issues manifest themselves in data gathered through *external, active* monitoring. We introduce a taxonomy of root causes in server performance degradations, which serves as the basis for our experiments. Furthermore, we describe the methods and steps we take to obtain our results and explain how the results will be examined and discussed. Following this, we will outline the design of the simulations that will be conducted, as well as the physical experimental setup enabling the simulations. We conclude by providing interpretation of the simulation based results, explaining how we can derive certain conclusions based on exploratory visual analysis and statistical changepoint analysis.

2 Root Causes of Server Performance Degradation

In general, if we consider performance and computation power of a system, we must consider resources that enable computation. These are usually hardware resources, such as processors, memory, disk I/O, and network bandwidth. We also need to consider the ways and methods these resources are allocated and utilized. The demand on resources of a computer system increases as the workload for the application of interest increases. When the demand of the application is greater than the resources that can be supplied by the underlying system, the system has hit its resource constraints. This means the maximum workload of the application has been reached and, typically, the time taken for each request to the application will increase. In case of extreme oversaturation, the system stops reacting entirely. For Web applications and Web services, this translates into poor response times or (temporary) unavailability. A delay in performance as observed through active monitoring can be defined as a negative change in response time at a certain point in time t. This means we look at two observations of response times x_t and x_{t+1} where $\lambda = |x_t - x_{t+1}| > c$ and c denotes a certain threshold accounting for possible volatility. In the following, this simplified notion of performance delays λ over a threshold c will be used in the description of the elements of the taxonomy.

The underlying causes of performance degradations in Web application and Web service backends are diverse. They differ significantly in the way they manifest themselves in performance data gathered through active monitoring. We propose a simple taxonomy of root causes in Web performance degradation. First, we divide a possible root cause in three main categories, which can be determined through external monitoring: *global delay*, *partial delay*, and *periodic delay*. Further classifications and proper assignment to the main categories in the taxonomy have been derived together with domain experts in the area of Web performance monitoring and optimization. In the following, we provide a brief explanation of the general causes of performance delays in computer systems. We then classify the main categories of our taxonomy by grouping the root causes by the distinguishable effect they have. The taxonomy is depicted in Figure 1, and explained in more detail the following. Note that this taxonomy does by no means raise the claim of completeness. It is rather an attempt to give an overview of common pitfalls that cause slowness in performance.

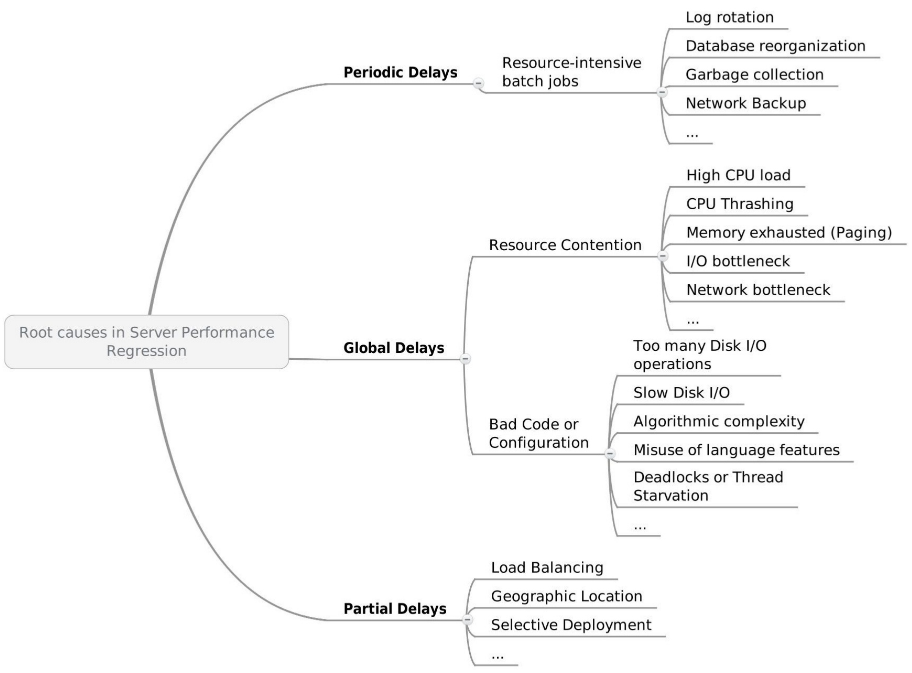


Fig. 1. Taxonomy of Root Causes in Server Performance Degradation

2.1 Global Delay

A global delay means that a change in a new deployment or release of the backend system introduced a significant difference in response time λ, which is higher than a defined threshold c on *all* incoming requests. We distinguish between global delays that are caused through resource contention and code or configuration issues. Global delays caused by resource contention include, for instance, delays due to a bottleneck in disk I/O. In this case, the flow of data from the disk to the application (e.g., via a database query) is contended. This means the query takes longer to perform and return data which causes an overall performance delay. Global delays caused by problems in the application code can for instance be caused by the induction of logical units or algorithms with high computational complexity in the backend service. Alternatively, such global delays may be caused by overzealous synchronization in the application code, which may even lead to temporary service outages when a deadlock situation cannot be immediately resolved by the underlying operating system or virtual machine.

2.2 Periodic Delay

Periodic delays are global delays that are not continuous, but happen, e.g., a few times a day. A periodic delay is mostly not induced through a new deployment or release, but rather through a background process causing the system to use an increased amount of resources. One practical example of such a background job is log rotation. Log rotation is an automated process in server systems administration, where log files that exhibit certain characteristics are archived. The process is usually configured to run as a periodic cronjob to be fully automated. Log rotating often includes archiving and transferring large text files, and, hence, can oversaturate the backend system for a short period of time.

2.3 Partial Delay

Partial delays are global delays that occur only for a subset of all requests, e.g., for a subset of all customers of the Web service. This situation can occur if the application employs a form of redundancy, e.g., load balancing or content distribution networks. In such scenarios, any problems that lead to global delays can potentially be inflicting only one or a subset of all backend servers, hence delaying only those requests that happen to be handled by one of the inflicted backends. Partial delays are interesting, as they are hard to detect (especially if the number of inflicted backends is small).

3 Identifying Root Causes of Performance Degradation

After introducing externally visible classes of root causes of performance degradation (global, partial, periodic), we want to identify characteristics in performance data associated with each class. Furthermore, we want to present statistical methods that are well suited for identifying such changes. Our approach is to generate realistic performance traces for each class through a testbed Web service, which we have modified in order to be able to inject specific *performance degradation scenarios* associated with each class. We collect data through active monitoring as described in Section 3.1. The specific scenarios we used are described and defined in Section 3.2. Afterwards, we apply the methods described in Section 4.1 to the traces we generated, in order to be able to make general statements about how these methods are able to detect root causes of performance degradation.

3.1 Simulation Design

We consider a simulation model of a simple Web service environment for our experiments. The model consists of the following components: *Synthetic Agent Nodes*, *Scenario Generation Component*, and *Dynamic Web Service Component.* Synthetic agents send out HTTP GET requests every n minutes from m agents and collect response times. In the simulation, we sample every 1 minute resulting in 1 new observation of our system every minute. Each observation is stored in a database with the corresponding timestamp.

The simulation design and its communication channels are depicted in Figure 2. We consider a system with this architecture where requests incoming from the synthetic nodes are governed by a stochastic process $\{Y(t), t \in T\}$, with T being an index set representing time.

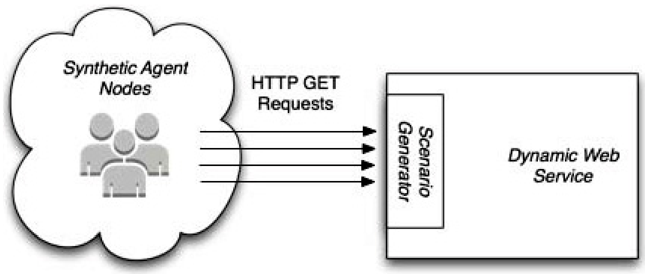


Fig. 2. Simulation Design

Synthetic Agent Nodes. We gather data from the *Dynamic Web Service Component* via active, periodic monitoring through *Synthetic Agent Nodes.* A synthetic monitoring agent acts as a client in order to measure availability and performance metrics, such as response time. Every synthetic agent is able to perform active measurements or synthetic tests. An active measurement is a request to a target URL, where subsequently all performance metrics that are available through the response are obtained. When configuring a set of synthetic tests, we can configure the following parameters: (1) URL of the Web service that should be tested; (2) sampling interval, e.g., every n minutes; (3) test duration, i.e., how many sample requests to issue in each run of the test agent.

Scenario Generation Component. As only the main classifications of root causes can be identified through synthetic monitoring, changes following the primary notions of *global delays*, *partial delays*, and *periodic delays* (see Section 2) are injected in the simulation. We achieve this by introducing a *Scenario Generation Component* into our model. It functions as an intermediary between the request sent by the synthetic agent nodes and the Web service. Instead of manually injecting faults into our test Web server, we define a set of scenarios that, subsequently, reflect the desired scenario, i.e., the faults over time in our system. The scenarios need to reflect performance degradation and performance volatility within a certain system, i.e., a single Web server. The Scenario Generation Component also needs to take into account possible geographic distribution of the agents, as well as load balancing mechanisms. In the following, we introduce a formal model for defining scenarios that reflects these notions that can be used to formally describe scenarios.

We consider a given set of parameters to compose a complete scenario within our simulation model. Within a scenario, we need to be able to specify how performance metrics (i.e., response times) develop over time, as well as synthetic agents that are actively probing our target system.

– A development $D \in \mathcal{D}$ maps from a certain point in $t \in T$ of the stochastic process $\{Y(t), t \in T\}$ (driving the requests of the synthetic agent nodes) to an independent random variable $X_i \in \mathcal{X}$, where $\mathcal{X}$ being the set of possible random variables (Equation 1).

$$D : T \mapsto \mathcal{X} \tag{1}$$

where $X_i \in \mathcal{X}$ and $\forall\ X_i \sim \mathrm{U}(a, b)$

– On the top-level, we define a scenario $\mathcal{S}$ that describes how the target system that is observed by each synthetic agent $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ develops over time. Each agent observes a development in performance $D_i \in \mathcal{D}$, with $\mathcal{D}$ being the set of all possible developments (Equation 2).

$$\mathcal{S} : \mathcal{A} \mapsto \mathcal{D} \tag{2}$$

This formalization allows us to express any performance changes (either positive or negative) as a classification of performance developments over time attributed to specific synthetic agents. More accurately, it models a performance metric as a uniformly distributed random variable of a system at any given time point $t \in T$. Specifying an assignment for every point in time is a tedious and unnecessary exercise. In order to define scenarios in a more efficient and convenient way, we introduce the following notation for developments $D \in \mathcal{D}$:

– Simple Developments: $[X_0, t_1, X_1, ..., t_n, X_n]$ defines a *simple development* as a sequence of independent random variables X_i and points in time t_i, further defined in Equation 3.

$$[X_0, t_1, X_1, ..., t_n, X_n](t) = \begin{cases} X_0 & 0 \leq t < t_1 \\ X_1 & t_1 \leq t < t_2 \\ \vdots & \\ X_n & t_n \leq t \end{cases} \tag{3}$$

This allows us to easily define developments X_i in terms of time spans $t_{i+1} - t_i \geq 0$ within the total time of observation. The last development defined through X_n remains until the observation of the system terminates.

– Periodic Developments: A *periodic development* is essentially a simple development, which occurs in a periodic interval p. It is preceded by a "normal phase" up until time point n. The "periodic phase" lasts for $p-n$ time units until the development returns to the "normal phase". A periodic development $[X_0, n, X_1, p]^*$ is defined in Equation 4.

$$[X_0, n, X_1, p]^*(t) = \begin{cases} X_1 & \text{for } kp + n \leq t < (k+1)p \\ X_0 & \text{otherwise} \end{cases} \tag{4}$$

where $k \geq 0$.

Figure 3 depicts how a periodic development can be seen over time with given parameters X_0 as the "normal phase" random variable, X_1 as the "periodic phase" random variable, n to define the time span for a "normal phase", p as the periodic interval and $(p - n)$ as the time span for the "periodic phase".

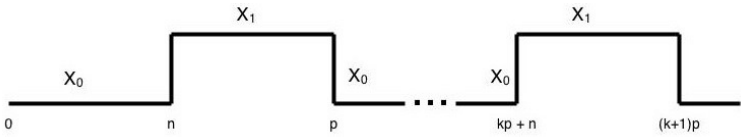


Fig. 3. Depiction of the periodic development scheme defined in Equation 4

These two defined functions allow us to conveniently define scenarios which adhere to our notions of global, partial and periodic delays. We can now define changes in performance at certain points in time in a declarative way, as well as define changes in periodic time intervals that result in changes in response times for a specific amount of time.

Dynamic Web Service Component. The *Dynamic Web Service Component* works together with the *Scenario Generation Component* to achieve the reflection of performance issues for the *Synthetic Agent Nodes*. In order to do so, it offers an endpoint that simulates delays over parameters passed from a specific scenario script. This means that the declared scenarios are executed within the Web service component and thus simulate workload through the parameters given in the scenarios.

3.2 Simulation Scenarios

Here, we formally define the parameters that actually form a certain scenario, and give an example of a real-life situation that would lead to such a performance behavior. The aim of each scenario is to properly represent one of global, partial or periodic delay.

Global Delay. A global delay is the introduction of a significant difference in response time on all incoming requests, i.e., it affects all users accessing the resource in question.

Example Scenario Use Case. A new feature needs to be implemented for a new release. A junior developer in charge of the new feature introduces a new (slow) database query, causing significantly higher overall response times. The slow query is not caught in QA (Quality Assurance) and the new release is deployed to all users.

Scenario Parameters. The parameter for this delay is given in Equation 5. Further, for every index i, we define the initial response time range as in Equation 6, as well as the range for the global change over all agents in Equation 7.

$$\mathcal{S}_G = \{a_i \mapsto [X_{a_i,0}, 420, X_{a,1}] \mid a_i \in \{a_1, a_2, a_3, a_4, a_5\}\} \tag{5}$$

$$\begin{aligned} X_{a_1,0} \sim \mathrm{U}(90,115), X_{a_2,0} \sim \mathrm{U}(100,130), X_{a_3,0} \sim \mathrm{U}(110,140), \\ X_{a_4,0} \sim \mathrm{U}(95,110), X_{a_5,0} \sim \mathrm{U}(100,110) \end{aligned} \tag{6}$$

$$X_{a,1} \sim \mathrm{U}(150,175) \tag{7}$$

Partial Delay. A partial delay scenario consists of requests that, at some point in time, cause a delay on a subset of the incoming requests.

Example Scenario Use Case. A Web application sits behind a load balancer handling 5 servers. One of the servers encounters unexpected hardware issues, which result in higher response times. The balancer uses "Round Robin" as its load balancing algorithm [22]. 20% of all users perceive the application with higher response times.

Scenario Parameters. The parameter for this delay is defined in Equation 8. For every index i we define the initial response time range as in Equation 9, as well as the range for the partial change for agent a_5 in Equation 10.

$$\mathcal{S}_P = \{a_i \mapsto [X_{a_i,0}, \infty] | a_i \in \{a_1, a_2, a_3, a_4, a_5\}, a_5 \mapsto [X_{a_5,0}, 360, X_{a_5,1}]\} \tag{8}$$

$$X_{a_1,0} \sim \mathrm{U}(115,125), X_{a_2,0} \sim \mathrm{U}(115,120), X_{a_3,0} \sim \mathrm{U}(120,145),\\ X_{a_4,0} \sim \mathrm{U}(105,115), X_{a_5,0} \sim \mathrm{U}(110,120) \quad (9)$$

$$X_{a_5,1} \sim \mathrm{U}(140,165) \quad (10)$$

Periodic Delay. A periodic delay takes place when, due to a (background) process, resource contention occurs and, subsequently, higher usage of hardware resources leads to higher response times for a certain amount of time. This scenario addresses those processes that are (usually) planned ahead and are executed within a specific interval.

Example Scenario Use Case. Log files of an application make up a large amount of the server's disk space. The system administrator creates a cron job to process older log files and move them over the network. The process induces heavy load on CPU (processing) and I/O (moving over network), which result into temporarily higher response times. The cron job is configured to take place in periodic intervals to ensure the server has enough disk space.

Scenario Parameters. The parameter for this periodic delay is defined in Equation 11. The response time ranges are defined as in Equation 12.

$$\mathcal{S}_{PD} = \{a_1 \mapsto [X_0, 45, X_1, 65]^*\} \quad (11)$$

$$X_0 \sim \mathrm{U}(95,115), X_1 \sim \mathrm{U}(160,175) \quad (12)$$

4 Experiments

We now describe the methods of analysis and execution of the experiments introduced in Section 3, as well as the concrete results we achieved.

4.1 Methods of Analysis

The specified scenarios are executed within a physical testbed (described in Section 4.2) and results are analyzed and interpreted. The following sections outline the methods of analysis that are applied to the results.

Exploratory Data Analysis. Our first analysis approach is to examine the time series of monitored response times over time visually over graphs in order to explore and gain further understanding on the effect specific underlying causes have on the resulting data. We also determine what kind of statistical attributes are well suited to identify key characteristics of the data sample and for humans to properly observe the performance change. For this initial analysis we plot the raw time series data[1] as line charts. This allows for visual exploratory examination, which we further complement with proper interpretations of the data displayed in the visualization that correlates with induced changes in the server backend.

Statistical Analysis. After manually inspecting the time series over visual charts, we evaluate the observation of performance changes by the means of statistical changepoint analysis. Specifically, we evaluate algorithms that are employed in the R [20] package "changepoint". This approach makes sense, as we are not interested in the detection of spikes or other phenomena that can be considered as outliers in the statistical sense, but rather in the detection of fundamental shifts in our data that reflect a longer standing performance degradation. We also want to keep false positives low, and determine whether a change is actually a change that implies a root cause that requires some kind of action. Thus, we also need to determine the magnitude of the change. For this, we recall the simple view on delays we introduced in our taxonomy: $\lambda = |x_t - x_{t+1}| > c$. We adapt this model of change as follows. Instead of comparing two consecutive data points x_t and x_{t+1}, we compare the changes in the mean at the time point where changepoint have been detected by the algorithm. In other words, we compute the difference between the mean of the distribution before the detected changepoint occurred, $\mu_{<\tau}$, and the mean of the distribution after the detected changepoint occured, $\mu_{>\tau}$, where τ denotes the changepoint. This difference is then denoted as λ, and can be defined as $\lambda = |\mu_{<\tau} - \mu_{>\tau}| > c$ via replacing two variables.

The threshold c is challenging to determine optimally. When c is set up too high, legitimate changes in performance that were caused by problems may not be detected and the system is in risk of performance degradation. When c is defined unnecessarily sensitive, the monitoring system is prone to false positives. The value of the threshold depends on the application and must be either set by a domain expert or be determined by statistical learning methods through analysis of past data and patterns. Sometimes it is useful to compare new metrics in relation to old metrics, this is a very simple way of statistical learning through past data. In the conducted experiments, we set the threshold as $c = \mu_{<\tau} \cdot 0.4$. This means that if the new metric after the changepoint $\mu_{>\tau}$ is 40% above or below the old metric $\mu_{<\tau}$, the change is considered a *real change* as opposed to a *false positive*. If we want to consider positive changes as well, the calculation of the threshold must be extended in a minor way to not yield into false negatives due to a baseline that is too high: $c = \min(\mu_{>\tau}, \mu_{<\tau}) \cdot 0.4$.

Note that, in the case of this paper, the threshold 40% of the mean was chosen after discussions with a domain expert, as it is seen as an empirically estimated

[1] Raw in this context means that we will not smooth the data by any means and will not apply statistical models in any way.

baseline. In practice, a proper threshold depends on the type of application, SLAs, and other factors and is usually determined by own empirical studies.

4.2 Testbed Setup

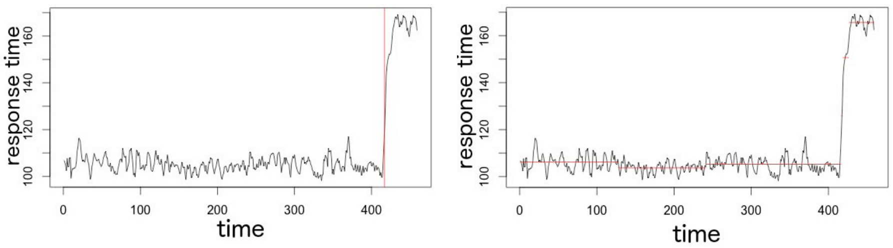
The experiments based on the described simulation model were executed in a local testbed consisting of 5 *Synthetic Agent Nodes* and 1 *Dynamic Web Service Component*. The agents operate in the local network as well. Each of the 5 nodes sends out a request every 5 minutes that is handled by a scheduler that uniformly distributes the requests over time. This results into 1 request/minute that is being send out by an agent that records the data. The physical node setup consists of Intel Pentium 4, 3.4 GHz x 2 (Dual Core), 1.5 GB RAM on Windows and is running a Catchpoint agent node instance for monitoring. The Web service running on the target server has been implemented in the Ruby programming language and runs over the HTTP server and reverse proxy nginx and Unicorn.The physical web server setup is the same as the synthetic agent node setup, but is running on Ubuntu 12.04 (64 bit). During simulation, a random number generator is applied to generate artificial user behavior as specified in the distributions, which can be represented efficiently with common random numbers [12]. However, as with deterministic load tests, replaying user behavior data may not always result into the same server response. Even with the server state being the same, server actions may behave nondeterministically. To adhere the production of independent random variables that are uniformly distributed we use MT19937 (Mersenne twister) [18] as a random number generator.

4.3 Results and Interpretation

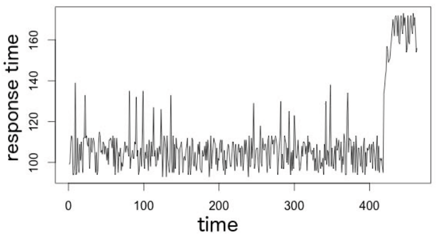
We now discuss the results of our scenario data generation, and the results of applying the statistical methods described in Section 4.1. At first, we display a raw plot of the resulting time series data without any filters and interpret its meaning. Further, we apply a moving average smoothing filter (with a window size $w = 5$) to each resulting time series and conduct a changepoint analysis.

Global Delay. The global delay forms the basis of our assumptions on how performance changes can be perceived on server backends. Both, the partial and periodic delay, are essentially variations in the variables time, interval and location of a global delay. Hence, the findings and interpretations of this section on global delays are the foundation for every further analysis and discussion.

Exploratory Visual Analysis. In Figure 4c, we see the results for the global delay scenario. We can clearly see the fundamental change in performance right after around 400 minutes of testing. The data before this significant change does seem volatile. There are a large amount of spikes occurring, though most of them seem to be outliers that might be caused in the network layer. None of the spikes sustain for a longer period of time, in fact between the interval around 150 and 250 there seem to be no heavy spikes at all. The mean seems stable around 115ms in response time and there is no steady increase over time that might suggest higher load variations. Thus, we can conclude that the significant performance change has occurred due to a new global release deployment of the application.



(a) Variance Changepoint (b) Mean Changepoint



(c) Global delay results raw time series

Fig. 4. Global Delay results

Statistical Changepoint Analysis. We apply changepoint analysis to the smoothed time series with a moving average window size of 5. In Figure 4a, we can immediately see how the smoothing affected the chart, compared to the chart with the raw data in Figure 4c: The spikes, i.e., the random noise, have been canceled out to a certain extent, making the main signal stronger and easier to identify. This makes it easier for our statistical analysis to focus on our signal and detect the proper underlying distributions. While this is definitely a pleasant effect of every smoothing technique, we also need to keep in mind that every model that we apply contains its own assumptions and own errors that need to be considered. What can further be seen in Figure 4a is the changepoint in the variance, denoted by a vertical red line at the changepoint location. Table 1 contains the numerical results of the changepoint in variance analysis and indicates the estimation for the changepoint τ_1 at 422. This number coincides approximately with our own estimation we concluded in the previous section.

Next, we look at Figure 4b, where the change in the mean is indicated by horizontal lines depicting the mean value for each segment that has been detected. This method has detected more changepoints than the previous analysis, which can be not clearly seen in Figure 4b due to the very small change in the mean. The estimated numerical values for the changepoints are listed in Table 2.

The table also lists the mean values, as well as the calculated threshold $c = \mu_{<\tau} \cdot 0.4$. The last column also states whether or not a detected changepoint in the mean is an *actual changepoint* (CP) as defined by the notion of *significant change* where $\lambda = |\mu_{<\tau} - \mu_{>\tau}| > c$, or a *false positive* (FP). Only one of the estimated changepoints has been identified as CP when considering the threshold c. This shows that detecting a fundamental change is a difficult undertaking, especially considering non-parametric statistical analysis, as in our case. Post-processing and analysis of the estimated changepoints and its according mean values is important to avoid false positives.

Table 1. Variance CP for $\mathcal{S}_G$

τ	$\sigma^2_{<\tau}$	$\sigma^2_{>\tau}$
422	10.695	67.731

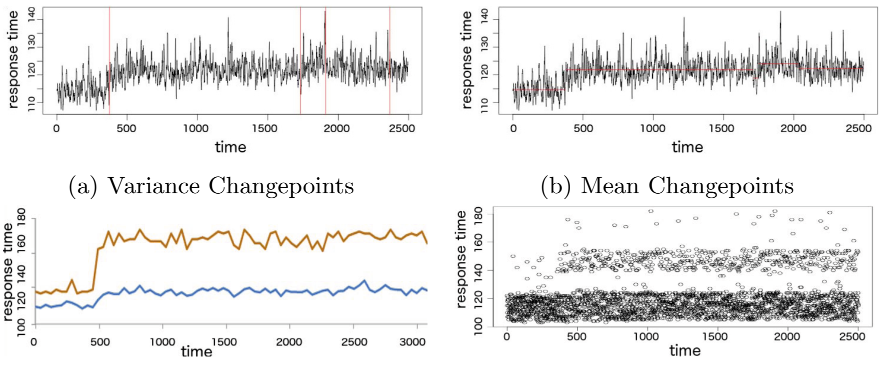
Table 2. Mean CP for $\mathcal{S}_G$

τ	$\mu_{<\tau}$	$\mu_{>\tau}$	λ	c	CP/FP
127	106.18	103.7	2.48	42.47	FP
241	103.7	105.32	1.62	41.48	FP
366	105.32	110.85	5.53	42.12	FP
374	110.8	103.62	7.23	44.32	FP
421	103.62	150.65	47.03	41.44	**CP**
427	150.65	165.62	14.97	60.62	FP

Partial Delay. Partial delays are global delays that only occur on a certain subset of requests and, therefore, need different techniques to properly examine the time series data and diagnose a performance degradation. Experiments on simulating partial delays have found that detection of a changepoint in partial delays, or even the visual detection of performance change, is not at all trivial.

Exploratory Visual Analysis. As before, we plot the time series data and look for changes in our performance. In Figures 5a and 5b, we see rather stable time series charts, relatively volatile (spiky), due to the higher amount of conducted tests and random network noise. Around the time points 460-500 and 1600-1900, we see a slight shift, but nothing alarming that would be considered a significant change. From this first visual analysis, we would probably conclude that the system is running stable enough to not be alerted. However, that aggregation hides a significant performance change. The convenience, or sometimes necessity, of computing summary statistics and grouping data to infer information this time concealed important facts about the underlying system. In order to detect this performance change, we have to look at additional charts and metrics.

In Figure 5c, we plot the same aggregation as in Figures 5a and 5b, but also plot the 90th percentile of the data to come to a more profound conclusion: There actually has been a performance change that is now very clear due to our new plot of the 90th percentile. While this can mean that percentiles also show temporary spikes or noise, it also means that if we see a significant and persisting shift in these percentiles, but not in our average or median, that a subset of our data, i.e., a subset of our users, indeed has experienced issues in performance and we need to act upon this information. Another way of detecting issues of this kind is to plot all data points in a scatterplot. This allows us to have an overview of what is going on and to identify anomalies and patterns more quickly. As we can see in Figure 5d, a majority of data points is still gathered around the lower response time mean. But we can also see clearly that there has been a movement around the 400 time point mark that sustains over the whole course of the observation, building its own anomaly pattern.



(a) Variance Changepoints (b) Mean Changepoints

(c) 90th percentile and mean plot (d) Partial Delay Scatterplot

Fig. 5. Partial Delay results

Statistical Changepoint Analysis. Analyzing both the changepoints in the variance in Table 3 and Figure 5a, as well as the changepoints in the mean in Table 4 and Figure 5b yields no surprise following our initial exploratory visual analysis. The changes that have been identified are not significant enough to be detected through the mean and the variance respectively. Although we need to point out that both analyses actually detected the actual significant changepoint around the time point 374-376, but are disregarded as false positives by our post-processing step of checking the threshold. Thus, our post-process actually resulted into a *false negative.* This is usually a sign that an indicator (in our case the threshold c) needs to be adjusted or rethought completely. However, before this kind of decision can be made, more information has to gathered on how this indicator has performed in general (i.e., more empirical evidence on false negatives, ratio between false positives and false negatives, etc.).

In order for our regular statistical analysis process, as applied previously, to properly work we need a further pre-processing step. Neither mean nor variance can detect the performance change, therefore, we need to consider a different metric. In our exploratory analysis, we concluded that the 90th percentile was able to detect the change. Thus, we need to apply our changepoint analysis to percentiles in order to detect a significant shift for partial delays.

Table 3. Variance CP for $\mathcal{S}_P$

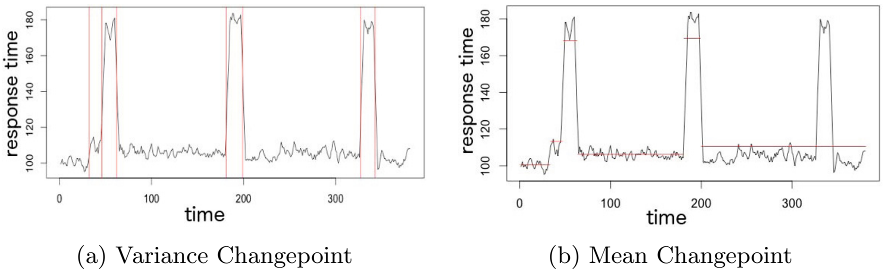
τ	$\sigma^2_{<\tau}$	$\sigma^2_{>\tau}$
374	12.88	24.25
1731	24.25	12.43
1911	12.43	6.63

Table 4. Mean CP for $\mathcal{S}_P$

τ	$\mu_{<\tau}$	$\mu_{>\tau}$	λ	c	CP/FP
376	114.74	121.92	7.18	45.88	FP
1710	121.92	118.91	3.01	48.76	FP
1756	118.91	124.1	5.19	47.56	FP
2035	124.1	122.27	1.83	49.64	FP

Periodic Delay. Periodic delays are either global or partial delays that occur in specific intervals, persist for a certain amount of time, and then performance goes back to its initial state.

Exploratory Visual Analysis. For this experiment we recorded enough values to result into three periods that indicate a performance degradation for a certain amount of time before returning back the system returns to its normal operation. The intuition we have on periodic delays can be clearly observed in Figure 6. Between the phases, we have usual performance operation with usual volatility, and in between we see fundamental shifts that persist for approximately 20 minutes before another shift occurs. Seeing periodic delays is fairly simple, as long as we are looking at a large enough scale. If we would have observed the time series within the periodic phase, before the second shift to the normal state occurred, we might have concluded it to be a simple global delay. Thus, looking at time series at a larger scale might help to identify periodic delays that would have otherwise been disregarded as short-term trends or spikes.



(a) Variance Changepoint (b) Mean Changepoint

Fig. 6. Periodic Delay results

Statistical Changepoint Analysis. While the exploratory visual analysis in periodic delays was straight forward, the changepoint analysis brings some interesting insights. Figure 6a and Table 5 show the analysis for the changepoints in the variance, which mostly coincide with our visual assessment. Merely the first detected changepoint in the variance is a false negative.

The changepoint in the mean yields more interesting results, as seen in Table 6 Contrary to the results in the other experiments the number of false positives is very low at only one. When looking at the result in Figure 6b, we can observe another interesting phenomena we have not seen before, a false negative. The third period was not detected as a changepoint in the mean by the algorithm, although it was detected by the changepoint in the variance method. This means, in order to avoid false negatives, we should apply both analyses for changepoint in the mean and variance.

Table 5. Variance CP for $\mathcal{S}_{PD}$

τ	$\sigma^2_{<\tau}$	$\sigma^2_{>\tau}$
32	5.21	20.22
46	20.22	156.25
62	156.25	11.73
181	11.73	233.96
199	233.96	16.3
327	16.3	180.3
343	180.3	25.16

Table 6. Mean CP for $\mathcal{S}_{PD}$

τ	$\mu_{<\tau}$	$\mu_{>\tau}$	λ	c	CP/FP
33	100.47	113.27	12.8	40.18	FP
47	113.27	168.11	54.84	45.31	**CP**
63	168.11	106.18	61.93	42.47	**CP**
180	106.18	169.35	63.17	42.46	**CP**
199	169.35	110.52	58.83	44.20	**CP**

4.4 Threats to Validity

In this research, we employ a simplified simulation model in order to show how significant performance changes can be detected in continuously observed data of a constructed target system. The creation and design of a simulation model comes with inherent risk concerning the validity of the results when applied to real-life systems. Most importantly, in our simulation design, we do not actively inject the notion of web traffic workloads (as has been, for instance, taken in consideration in [16]), but rather simulate the variability of workloads (and therefore response times) in web services through specific scenarios parameters. Further, the notions of network traffic and network volatility in wide area networks (WAN) are completely omitted to limit the interference of network noise in the recorded data. Of course, noise is still present even in LANs, but is limited to very few factors within the data center, as opposed to the possible factors in WANs. This also means that there is, for instance, no increased response time due to DNS resolution in our experiments.

5 Related Work

Analyzing performance data observed through continuous monitoring in distributed systems and web services has produced a large body of research dealing with statistical modeling of underlying systems, anomaly detection and other traditional methods in statistics to address problems in performance monitoring and analysis. For instance, Pinpoint [8] collects end-to-end traces of requests in large, dynamic systems and performs statistical analysis over a large number of requests to identify components that are likely to fail within the system. It uses a hierarchical statistical clustering method, using the arithmetic mean to calculate the difference between components by employing the Jaccard similarity coefficient [13]. [11] makes use of statistical modeling to derive signatures of systems state in order to enable identification and quantification of recurrent performance problems through automated clustering and similarity-based comparisons of the signatures. [1] proposes an approach which identifies root causes of latency in communication paths for distributed systems using statistical techniques. An analysis framework to observe and differentiate systemic and anomalous variations in response times through time series analysis was developed in [9]. A host of research from Borzemski *et al.* highlights the use of statistical methods in web performance monitoring and analysis [2–6]. The research work spans from statistical approaches to predict Web performance to empirical studies to assess Web quality by employing statistical modeling. [19] suggests an approach to automated detection of performance degradations using control charts. The lower and upper control limits for the control charts are determined through load testing that establish a baseline for performance testing. The baselines are scaled employing a linear regression model to minimize the effect of load differences. [10] proposes an automated anomaly detection and performance modeling approach in large scale enterprise applications. A framework is proposed that integrates a degradation based transaction model reflecting resource consumption and an application performance signature that provides a model of runtime behavior. After software releases, the application signatures are compared in order to detect changes in performance. [17] conducted an experimental study comparing different monitoring tools on their ability to detect performance anomalies through correlation analysis among application parameters. In the past, we have also applied time series analysis to the prediction of SLA violations in Web service compositions [15].

6 Conclusion and Future Work

A taxonomy of root causes in performance degradations in web systems has been introduced, which was further used to construct scenarios to simulate issues in web performance within an experimental setup. In a series of simulations, we measured how performance metrics develop over time and presented the results. Furthermore, we provided analysis and interpretation of the results. Following the work presented in this paper, there are possible improvements and further work we were not able to address sufficiently. Performance data gathered through external synthetic monitoring only allows for a black box view of the system and is often not sufficient for in-depth root cause analysis of performance issues. Combining data from external monitoring and internal monitoring in order to automate or assist in root cause analysis and correlation of issues is a possible approach that should be considered. The simulation and analysis in this paper is limited to performance issues on the server. Further work might include extending the taxonomy of root causes and simulation scenarios to also represent frontend performance issues.

Acknowledgements. The research leading to these results has received partial funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement 610802 (CloudWave). Numerical simulation was carried out during a research visit of the first author at Catchpoint Systems, Inc., New York, USA. The authors would like to thank Prof. Gernot Salzer for his comments on earlier versions of this work.

References

1. Aguilera, M.K., Mogul, J.C., Wiener, J.L., Reynolds, P., Muthitacharoen, A.: Performance debugging for distributed systems of black boxes. ACM SIGOPS Operating Systems Review 37, 74–89 (2003)
2. Borzemski, L.: The experimental design for data mining to discover web performance issues in a wide area network. Cybernetics and Systems 41(1), 31–45 (2010)
3. Borzemski, L., Drwal, M.: Time series forecasting of web performance data monitored by MWING multiagent distributed system. In: Pan, J.-S., Chen, S.-M., Nguyen, N.T. (eds.) ICCCI 2010, Part I. LNCS, vol. 6421, pp. 20–29. Springer, Heidelberg (2010)
4. Borzemski, L., Kamińska-Chuchmała, A.: Knowledge discovery about web performance with geostatistical turning bands method. In: König, A., Dengel, A., Hinkelmann, K., Kise, K., Howlett, R.J., Jain, L.C. (eds.) KES 2011, Part II. LNCS, vol. 6882, pp. 581–590. Springer, Heidelberg (2011)
5. Borzemski, L., Kaminska-Chuchmala, A.: Knowledge engineering relating to spatial web performance forecasting with sequential gaussian simulation method. In: KES, pp. 1439–1448 (2012)
6. Borzemski, L., Kliber, M., Nowak, Z.: Using data mining algorithms in web performance prediction. Cybernetics and Systems 40(2), 176–187 (2009)
7. Bouch, A., Kuchinsky, A., Bhatti, N.: Quality is in the eye of the beholder: meeting users' requirements for internet quality of service. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 297–304. ACM (2000)
8. Chen, M.Y., Kiciman, E., Fratkin, E., Fox, A., Brewer, E.: Pinpoint: Problem determination in large, dynamic internet services. In: Proceedings of International Conference on Dependable Systems and Networks, DSN 2002, pp. 595–604. IEEE (2002)
9. Chen, Y., Mahajan, R., Sridharan, B., Zhang, Z.-L.: A provider-side view of web search response time. SIGCOMM Comput. Commun. Rev. 43(4), 243–254 (2013)
10. Cherkasova, L., Ozonat, K., Mi, N., Symons, J., Smirni, E.: Automated anomaly detection and performance modeling of enterprise applications. ACM Transactions on Computer Systems (TOCS) 27(3), 6 (2009)
11. Cohen, I., Zhang, S., Goldszmidt, M., Symons, J., Kelly, T., Fox, A.: Capturing, indexing, clustering, and retrieving system history. ACM SIGOPS Operating Systems Review 39, 105–118 (2005)
12. Heikes, R.G., Montgomery, D.C., Rardin, R.L.: Using common random numbers in simulation experiments - an approach to statistical analysis. Simulation 27(3), 81–85 (1976)
13. Jaccard, P.: The distribution of the flora in the alpine zone. 1. New Phytologist 11(2), 37–50 (1912)
14. King, A.: Speed up your site: Web site optimization. New Riders, Indianapolis (2003)

15. Leitner, P., Ferner, J., Hummer, W., Dustdar, S.: Data-Driven Automated Prediction of Service Level Agreement Violations in Service Compositions. Distributed and Parallel Databases 31(3), 447–470 (2013)
16. Liu, Z., Niclausse, N., Jalpa-Villanueva, C., Barbier, S.: Traffic Model and Performance Evaluation of Web Servers. Technical Report RR-3840, INRIA (December 1999)
17. Magalhaes, J.P., Silva, L.M.: Anomaly detection techniques for web-based applications: An experimental study. In: 2012 11th IEEE International Symposium on Network Computing and Applications (NCA), pp. 181–190. IEEE (2012)
18. Matsumoto, M., Nishimura, T.: Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation (TOMACS) 8(1), 3–30 (1998)
19. Nguyen, T.H., Adams, B., Jiang, Z.M., Hassan, A.E., Nasser, M., Flora, P.: Automated detection of performance regressions using statistical process control techniques. In: Proceedings of the Third Joint WOSP/SIPEW International Conference on Performance Engineering, pp. 299–310. ACM (2012)
20. R Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, (2013)
21. Forrester research. Ecommerce web site performance today: An updated look at consumer reaction to a poor online shopping experience (August 2009)
22. Shirazi, B.A., Kavi, K.M., Hurson, A.R. (eds.): Scheduling and Load Balancing in Parallel and Distributed Systems. IEEE Computer Society Press, Los Alamitos (1995)

Indexing Rich Internet Applications Using Components-Based Crawling

Ali Moosavi[1], Salman Hooshmand[1], Sara Baghbanzadeh[1],
Guy-Vincent Jourdan[1,2], Gregor V. Bochmann[1,2], and Iosif Viorel Onut[3,4]

[1] EECS - University of Ottawa
[2] Fellow of IBM Canada CAS Research, Canada
[3] Research and Development, IBM® Security AppScan® Enterprise
[4] IBM Canada Software Lab., Canada
{smousav2,shooshma,sbaghban}@uottawa.ca,
{gvj,bochmann}@eecs.uottawa.ca,
vioonut@ca.ibm.com

Abstract. Automatic crawling of Rich Internet Applications (RIAs) is a challenge because client-side code modifies the client dynamically, fetching server-side data asynchronously. Most existing solutions model RIAs as state machines with DOMs as states and JavaScript events execution as transitions. This approach fails when used with "real-life", complex RIAs, because the size of the produced model is much too large to be practical. In this paper, we propose a new method to crawl AJAX-based RIAs in an efficient manner by detecting "components", which are areas of the DOM that are independent from each other, and by crawling each component separately. This leads to a dramatic reduction of the required state space for the model, without loss of content coverage. Our method does not require prior knowledge of the RIA nor predefined definition of components. Instead, we infer the components by observing the behavior of the RIA during crawling. Our experimental results show that our method can index quickly and completely industrial RIAs that are simply out of reach for traditional methods.

Keywords: Rich Internet Applications, Web Crawling, Web Application Modeling.

1 Introduction

In the past decade, modern web technologies such as AJAX, Flash, Silverlight, etc. have given emergence to a new class of more responsive and interactive web applications commonly referred to as Rich Internet Applications (RIAs). RIAs make Web-applications more interactive and efficient by introducing client-side computation and updates, and asynchronous communications with the server [1]. Crawling RIAs is more challenging than crawling traditional web applications because some core characteristics of traditional web applications are violated by RIAs. Client states no longer correspond to unique URLs as modern web technologies enable the ability to change the client state and even populate it with new data without changing the URL, up to the point that it is possible to have complete complex web applications with a single URL. Moreover, JavaScript events (from here on, called *"events"*) can take the place of hyperlinks; and unlike hyperlinks, the crawler cannot predict the outcome of an event before executing it. In that sense, the behavior and user interface of a RIA is more similar to an event-driven software like a desktop software GUI than to a traditional web application.

The problem of crawling AJAX-based RIAs has been a focus of research in the past few years. In such RIAs, executing events can change the Document Object Model (DOM), hence leading the RIA to a new client state, and can possibly leaned to message exchanges with the server, changing the server state as well. A common approach is to model a RIA as a finite state machine (FSM). In the FSM, DOMs are represented as states and event executions are represented as transitions. Events can lead from one DOM-state[1] to another.

One simplifying assumption that is usually made is that server states are in sync with client states. Therefore by covering all client states (i.e. DOM-states), the crawler has also covered all server states. Based on this assumption, by executing each event from each DOM-state once and building a complete FSM model from the RIA, the crawler can assume that the RIA has been entirely covered and modeled. In order to stay in sync with server, however, the crawler cannot jump to arbitrary DOM-states at will (e.g. by saving DOM-state in advance and restoring it when desired); instead, it must follow a sequence of events. If the desired DOM-state is not reachable from the current DOM-state using a chain of transitions (called a "transfer sequence"), the crawler needs to issue a "reset" (reloading the URL of the initial page) to go to the initial DOM-state and follow a valid transfer sequence from there. Resets are usually modelled as special transitions from all DOM-states to the initial DOM-state. In the beginning, the only known DOM-state is the initial DOM-state and all its events are unexecuted yet. By executing an unexecuted event, the crawler discovers its destination, which might be a known DOM-state or a new one. The event execution can then be modelled as a transition between its source and destination DOM-states. A state machine can be represented as a directed graph. The problem of crawling a RIA is therefore that of exploring an unknown graph. At any given time, the crawler needs to execute an unexecuted event, or use the known portion of the graph to traverse to another DOM-state to execute one, until all events in the graph have been executed, at which point the graph is fully uncovered and crawling is done. Based on this model, different exploration strategies (such as depth-first search, Greedy and Model-Based strategies, see Section 2) have been suggested. Comparing different exploration strategies can be done by comparing the number of events and resets executed during the crawl. We call this the *"exploration cost"*.

[1] Related works in the literature commonly refer to DOM-states simply as "states". In this work we differentiate between "DOM-states" and what we will call "component-states".

One major challenge in this field is a state space explosion: the model being built grows exponentially in the size of the RIAs being crawled. This state space explosion not only leads to production of a large model that is difficult to analyze, but also makes the crawlers unable to finish in a reasonable time. Because of this excessive running times, all DOM-based methods that have been published so far are essentially unsuitable for real-life scenarios. These methods cannot be used in the industrial world. Current studies use different notions of DOM equivalence to map several DOMs to one state. These approaches usually involve applying reduction and normalization functions on the DOM. While these approaches have been used and tested successfully on experimental RIAs, they fail to provide satisfactory equivalency criteria when faced with real-world large-scale RIAs.

Most of the time, this state space explosion is caused by having the same data being displayed in different combinations, leading to large sets of new DOMs and large state space for a small set of functionalities. In a typical RIA, it is common to encounter a new DOM-state which is simply a different combination of already-known data (Figure 1). We call this situation *new DOM-state without new data*. Such DOM-states should be ideally regarded as already known. Today's complex RIA interfaces consist of many interactive parts that are independent from one another, and the Cartesian product of different content that each part can show easily leads to an exponential blow-up of the number of DOM-states. In the following, we call these independent parts *components*, and each of their values *component-state*. A fairly intuitive example is widget-based RIAs, in which various combination of contents that each widget can show creates a very large number of different DOM-states. Not all these DOM-states are of interest to the crawler. A content indexing crawler, for instance, needs to retrieve the content once and finish in a timely fashion. These "rehashed" DOM-states only prolong crawling while providing no new data. Figure 1 provides an example. This issue is not just limited to widgets, but is present in any independent part in RIAs down to every single popup or list item. Typical everyday websites such as Facebook, Gmail and Yahoo, and any typical RIA mail client, enterprise portal or CMS contain dozens if not hundreds of independent parts. Different combinations of these independent parts lead crawlers through a seemingly endless string of new DOM-states with no new data. A human user, on the other hand, is not confused by this issue since she views these components as separate entities, and in fact would be surprised if the behavior of one of these parts turns out to be dependent on another.

We observe that one major drawback inherent to all these methods is that they model client states of RIA at the DOM level. We propose a novel method to crawl RIAs efficiently by modeling in terms of states of individual sub-trees of the DOM that are deemed independent, which we call *components*. Our method detects independent components of RIA automatically, using the result of diffs between DOMs. By modeling at the component level rather than at the DOM level, the crawler is able to crawl complex RIAs completely (and in fact quickly) while covering all the content. The resulting end-model is smaller and therefore easier for humans to understand and for machines to analyze, while providing

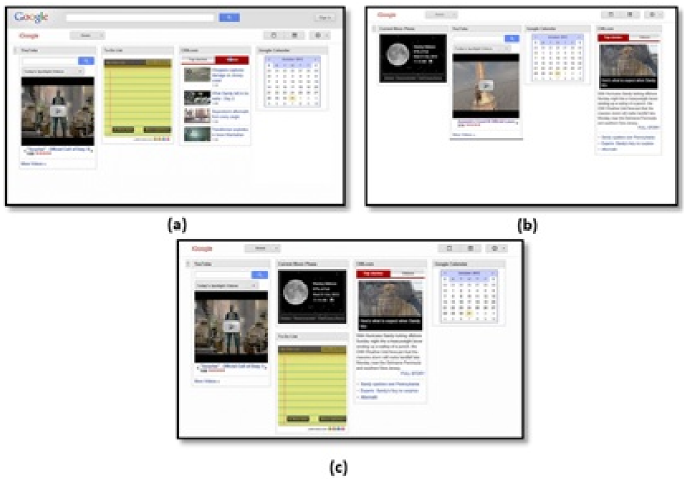


Fig. 1. Example of a new DOM-state with no new data. The DOM in **(c)** is only a combination of data already present in **(b)** and **(a)**, but will have a new DOM-state in the existing methods.

more information about the RIAs being modeled. As we will show in our experimental results, the method presented here is suitable to crawl and index real-life, complex RIAs, without loss of content in our experiments, where previously published methods failed to do the same thing even on much simplified version of the same RIAs.

The remainder of this paper is organized as follows. In Section 2, we provide a review of related work. In Section 3 we present the general overview of our solution. We first describe the model that the crawler builds, and we then describe how the crawler builds this model and makes use of it during the crawl. Experimental results and comparisons are presented in Section 4, and we conclude in Section 5.

2 Related Works

Crawling RIAs using a state transition model has been extensively studied. Duda et al. use a breadth-first search approach to explore RIA, assuming the ability to cache and restore client states at will [2, 3]. In [2], they point to the state space explosion problem caused by independent parts as an unresolved challenge. Amalfitano et al. use manual user-sessions to build a state machine [4]. In a follow-up work, they automate their tool by using depth-first exploration [5]. Peng et al. propose using a greedy algorithm as exploration strategy that outperforms depth-first and breadth-first search exploration significantly [6]. We use here the same greedy approach as exploration strategy. A different approach, called "model-based crawling" [7], focuses on finding the clients states as soon as possible by assuming some particular behavior from the RIAs [8,9]. The model used in [9] accumulates statical information about the result of previous event executions to infer what event to execute next. All the these method suffer from the problem of accumulating new DOM states that do no contain new data. In [10], an approach similar to model-based crawling is used, but for sites that have a known structure.

DynaRIA [11] provides a tool for tracing AJAX-Based application executions. It generates abstract views on the structure and run-time behavior of the application. The generated crawling model has been used for accessibility testing [12] or for generating test sequences [13]. It also also been used for modelling native android apps [14, 15] and native iOS apps [16].

All the above mentioned works use DOM-level state machines and use different DOM equivalence criteria to guide crawling: in [17], an edit distance between DOM-states is used. Methods based on DOM manipulations are used in [7–9,18]. These various DOM equivalence criteria do help but ultimately fail to address completely the state space explosion problem. To alleviate this problem, in [17] it is proposed to explore only new events that appear on a DOM after an event execution.This limits the crawler's ability to reach complete coverage, and does not prevent exploring redundant data when different event execution paths lead to the same structure (e.g. a widget frame) but in different DOMs. *FeedEx* [19] extends [6] by selecting states and events to be explored based on probability to discover new states and increase coverage. They include four factors to prioritize the events : code coverage, navigational diversity, page structural diversity and test model size. Surveys of RIAs crawling can be found in [20, 21].

In the context of detecting independent parts, static widget detection methods such as [22] and [23], and detection of underlying source dataset [24] have been developed. In [23], the use of patterns for detecting widgets based on static JavaScript code analysis and interaction between widget parts is proposed. However, these methods designed only to detect widgets or source datasets, which are a small subset of independent entities in RIAs.

3 Component-Based Crawling

3.1 Overview of Our Solution

Our solution is to model RIAs at a "finer" level, using subtrees of the DOM (called "components") instead of modeling in terms of DOMs. By building a state-machine at the component level, we get a better understanding of how the RIA behaves, which helps addressing the aforementioned state explosion problem [25]. The crawler can use this model along with its exploration strategy. Our prototype implementation uses the greedy algorithm presented in [26] as the exploration strategy. In this section we present a brief general overview of the concept of components, before providing more details in the following Sections.

In a typical real-life RIA such as the one depicted in Figure 2, a given "page" (DOM-state) contains a collection of independent entities. We call these entities "components". They are subtrees of the current DOM. Examples of components include menu bars, draggable windows in Twitter, as well as each individual tweet, chat windows in Gmail, as well as the frame around each chat window, the notifications drop-down and mouse-over balloons in Facebook, etc. Users normally expect to be able to interact with each of these components independently, without paying attention to the state of the other components on the page. Classical crawling methods do not consider these components, and consequently generate every possible combination of these components states while building the model. Our aim is to detect these components to crawl each of them separately. The assumption of independency between components enables our method to "divide and conquer" the RIA to overcome state space explosion without loss of coverage. We expect this assumption to hold true in almost all real-life RIAs as it follows human user intuition. We did not encounter any counterexamples in our investigation of real-life RIAs. If, however, there are components on a particular RIA that affect each other, the crawler might lose coverage of some of the content of the RIA since it does not analyze the interactions between the components. In our experiments, this situation did not occur.

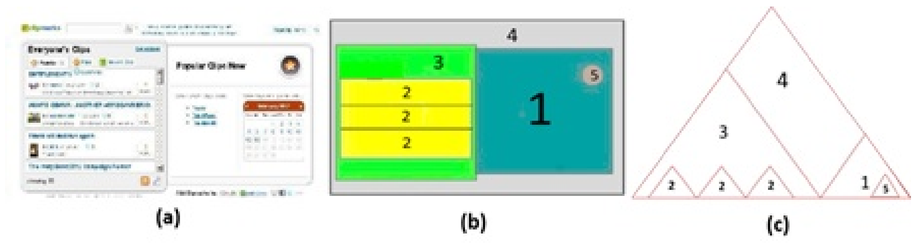


Fig. 2. **(a)** A webpage, **(b)** components on the page the way a human user sees them as entities of the page, and **(c)** the way the crawler sees them as subtrees of the DOM

The input of the crawler after each event execution is the DOM tree. Since components appear as subtrees in the DOM tree, we partition the DOM into multiple subtrees that are deemed independent of each other. Each of these subtrees correspond to a particular state of a component (a *component-state*). We model the RIAs as a combination of independent component states instead of assigning a DOM-state to the entire DOM. The idea of components and their associated component-states completely replace use of DOM-states in our method. Each component has a set of possible component-states, and a component-state of a particular component is only compared to other component-states of its own.

As explained, in our model, at any given time, the RIA is in a set of component-states, since it consists of different components each in its own component-state. It is worth mentioning that the DOM is partitioned into components in a collectively exhaustive and mutually exclusive manner, meaning that each XML-node on the DOM tree belongs to one and only one component. Modeling RIAs at the component level as several benefits. The most obvious one is that it reduces the state space by avoiding modeling separately every combination of component states, including the many instances in which the combination is new but each component state has been seen before (as depicted previously in Figure 2). Moreover, this fine-grained view of RIA helps the crawler map the effect of event executions more precisely, resulting in a simpler model of the RIA with fewer states and transitions. As a result, the crawler will traverse the RIA more efficiently by taking fewer steps when aiming to revisit a particular structure (such as a text or event) in the RIA that is not present in the current DOM. The resulting model of the RIA will also be more easily understandable by humans because it has fewer states and transitions and the effect of each event execution on the DOM is defined more clearly.

To illustrate the potential gain of our methods, imagine that the current DOM is made of k independent components $C_1, C_2, \ldots, C_k$. Assume that each component C_i has $\bar{C_i}$ components states. Using the traditional, DOM-state based method, this will lead to $\prod_{i=1}^{k} \bar{C_i}$ DOM-states. If in addition the components can be moved freely on the page, this number will be repeated $k!$ times, leading to $k! \prod_{i=1}^{k} \bar{C_i}$ DOM-states. This already intractable number will increase even more of some components are repeated or if some components can be removed from the DOM. Using our method yields only $\sum_{i=1}^{k} \bar{C_i}$ component-states for the same RIAs, even when the components are repeated or removed from the page.

3.2 Model Description

In our model, we partition each DOM into independent components. Each component has its own component-state so the current DOM corresponds to a set of component-states in the state machine. Because JavaScript events are attached to XML nodes, each event resides in one of the component-states present in the DOM. We call it the *"owner component-state"* of the event[2].

3.2.1 Multistate Machine

An event is represented as a transition that starts from its owner component-state. Since the execution of the event can affect multiple components, the corresponding transition can end in multiple component-states. Therefore, our model is a multi-state-machine. Figure 3 illustrates how an event execution is modeled in our method versus other methods. The destination component-states of a transition correspond to component-states that were not present in the DOM, and appeared as a result of the execution of the event.

Our model is a multistate-machine, defined as a tuple $M = (A, I, \Sigma, \partial)$ where A is the set of component-states, I is the set of initial component-states (those that are present in the DOM when the URL is loaded), Σ is the set of events, and $\partial : A \times \Sigma \rightarrow 2^A$ is a function that defines the set of valid transitions. Similarly to

[2] For events that are not attached to an XML-node on the DOM, such as timer events, a special global always-present component is defined as their owner component.

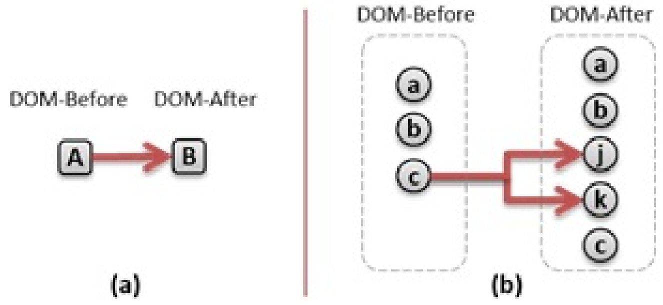


Fig. 3. An event execution modeled with **(a)** DOM-states, and **(b)** component-states. Rectangles in **(b)** represent DOM-states and are not used in the actual model.

the classical state-transition model, ∂ is a partial function, since not all events are available on all component states. Unlike the state-transition model, we have a set of initial states, and executing an event can modify any number of component-states. Our multistate-machine is resilient to shuffling components around in a DOM, and does not store information about exact location of the component-states in a DOM.

3.2.2 Components Definition

We have mentioned that components must be "independent" from one another. By this, we mean that the outcome of execution of an event only depends on the component-state of its owner component. In other words, the behavior of an events in a component is independent from the other components in the DOM and their individual component states. As an example, the border around a widget that has minimize/close buttons is independent of the widget itself, since it minimizes or closes regardless of the widget that it is displaying. Therefore, the widget border and the widget itself can be considered separate independent components. On the other hand, the next/previous buttons around a picture frame are dependent on that picture frame, since their outcome depends on the picture currently being shown. So the next/previous buttons should be put in the same component as the picture frame. Note that event executions outcome can affect any number of components and this does not violate the constraint of independency.

Two notions are important in our definition: first, we need to specify how we define a component, then how we capture the various component states that component might have.

Component are identified by an XPath, which specifies the root of the subtree that contains this component. In order to find a particular component in the DOM, one should start from the document root and follow the component's associated XPath. The element reached is the root of the component i.e. the component is the subtree under that element. Note that an XPath can potentially map to several nodes, therefore several instances of a component can be present in a DOM at the same time. Since the XPath serves as an identifier for a component, we need the XPath to be consistent throughout the RIA, i.e. it should be able to point to the intended subtree across different DOMs of the RIA. However, some attributes commonly used in XPath are too volatile to be consistent throughout the RIA. Hence, we only use the "id" and "class" attributes for each node in the XPath, and omit other predicates such as the position predicate.

Here is how we build an XPath for an element e: we consider the path p from the root of the DOM to e, and for each HTML element in p, we include the tag name of the element, the *id* attribute if it has one, and the *class* attribute if it has one.

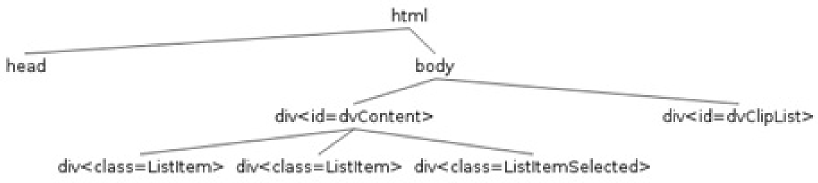


Fig. 4. Part of a shopping website's DOM

Figure 4 is an example of DOM for a shopping website. Individual list items in the product list are instances of a component "product list item". In this example, there are two instances of this component, which is identified by the XPath */html/body/div[@id="dvContent"]/div[@class="ListItem"]*. But the selected item in the list yields a different XPath since it is assigned a different class attribute, *[@class="ListItemSelected"]*.

Each component, identified by its XPath which points at the root of the component, can have a series of component states that are going to be uncovered throughout the crawl. These states are simply the content of the subtree found under the XPath. For efficiency reasons, we are not recording the entire subtree for each component state. Instead, we derive a unique identifier for the component state by hashing the content of the corresponding subtree[3]. In practice, the crawler keeps information on each component-state of each component in a data structure for use by the greedy algorithm. A simplified version of the data structure, called *stateDictionary*, is depicted in Table 1. In general, on any given DOM, some components are present in the DOM and some are not. Using the "Component Location" column, the crawler can find out which components are present on the DOM, then use the component's content to compute its ID.

[3] In reality, we first prune nested sub-components as explained later, and we also perform some transformations on the subtree to detect equivalents sub-components. See [27] for more details.

Using the “Component-State ID” column it can look up additional info on that component-state (event/transition destinations, unexecuted events, etc.), or discover that it is a new component-state.

Table 1. The StateDictionary

#	Component Location (XPATH)	Component State-ID (Hash)	Info
1	/	@J#F@)J#403rn0f29r3m19	....
2	/html/body/div[@id=”dvClipList”]	*&^$@J$$P@@$#$#_!$_*$_*	...
		GPDFJD}{PL”!{#R$$)%$*!_$#!!	...
3	/html/body/div[@id=”dvContent”]	VMLCVCPQ!#$!_()__IKEF)_I)	...
		{:$%@)(@#*GRJPGFD{#@)(	...
		?”$#%*%@$)(!#HI!_D}{|||#R!#!	...

Since components are identified by their XPath, it is possible to have nested components, with the root of one component (identified by its XPath) failing inside the subtree of another component. If a component contains other components, these components should be removed from the containing component when defining that component’s states. The pseudo-algorithm below explains how to list all the known component states that are present in the current DOM. Refer to [27] for more details.

Algorithm 1. Pseudo-code to find current component-states

```
procedure FindCurrentStates
    for all xpath in stateDictionary do
        ComponentInstances := go through the xpath and give the subtree
        for all Instance in ComponentInstances do
            for all sub-path under the current xpath do
                go through the sub-path and prune the subtrees
            end for
            stateID := ReadContentsAndComputeStateID(instance)
            Add the stateID to SetOfCurrentState
        end for
    end for
    return SetOfCurrentStates
end procedure
```

Note that the only location information for component states is an XPath to the root of the component state. Therefore, while our model is able to break a DOM into component-states, it is not possible to reconstruct an exact DOM using the multi-state-machine. While the resulting model of a RIA can be used to infer an execution trace to any given component state (thus any content of the RIA), it cannot be used to infer an execution trace to any given DOM.

3.3 Crawling Algorithm

As explained above, a crawling algorithm relies on an exploration strategy that tells it which events to execute next. Several exploration strategies can be used with our model. We explain our algorithm independently from the chosen strategy (which is “greedy” in our prototype).

Generally, using the “Component Location” list in the stateDictionary, the crawler can discover new component-states during the crawl and populate the “Component-State ID” lists. Our proposed component discovery algorithm populates the Component Location and Component-State ID lists incrementally during the crawl as it observes the behavior of the RIA. The algorithm is based on comparing the DOM tree snapshots before and after each event execution. Every time an event is executed by the crawler, the subtree of the DOM that has changed as a result of the event execution is considered a component.

The way we compare the DOM trees to obtain the changed subtree is defined as follows: suppose the DOM-tree before the event execution is T_{before} and the DOM tree after the event execution is T_{after}. We traverse T_{before} using breadth-first search. For each node x in T_{before}, we compute the path from root to x, and find the node y in T_{after} that has the same path. If x and y are different, or have different number of children, x is considered the root of a component; its XPath is added to the stateDictionary if not already existing, and the search is discontinued in the subtree of x . If several such nodes y exist in T_{after}, their deepest common ancestor is used as the root of the component.

Initially, the stateDictionary contains only one component with XPath “/”. Additional components are discovered and added to the stateDictionary as the crawling proceeds. The algorithm is summarized in the pseudo-code below. One important practical point to note is that the discovery of a new component can lead to a modification of previously known component states, if the new component is nested inside these component states. As explained before, the new component must be pruned from the containing component states, so their component state ID must be recomputed. It is not practical to save all component states DOM to be able to recompute their ID when this occurs. Instead, in our prototype we mark these component states as invalid and visit them again later during the crawl.

4 Experimental Results

4.1 Test Cases

In order to evaluate the efficiency of our method, we have run some crawling experiments with a number of experimental and real RIAs. We split these results in two categories. In the first category, we have seven simple RIAs[4]. Two of these are test application that we have built ourselves for testing purpose, while the five other ones are real, but simple (or simplified) RIAs. We have also run our

[4] `http://ssrg.eecs.uottawa.ca/testbeds.html`

Algorithm 2. Pseudo-code of proposed crawling algorithm

```
1: procedure ComponentBasedCrawl
2:     for as long as crawling goes do
3:         event := select next event to be executed based on the exploration strategy
4:         execute (event)
5:         delta := diff ($dom_{before}$, $dom_{after}$)
6:         xpath := getXpath (delta)
7:         if stateDictionary does not contain xpath then
8:             add xpath to stateDictionary
9:         end if
10:        resultingStates := FindCurrentStates(delta)
11:        for all state in resultingState do
12:            if stateDictionary does not contain state then
13:                add state to stateDictionary
14:            end if
15:            event.destinations := resultingStates
16:        end for
17:    end for
18:    return stateDictionary
19: end procedure
```

test on two real "complex" RIAs: IBM *Rational Team Concert* (RTC[5]), an agile application life-cycle management web-based application, and *MODX*[6], an open source content management system. For these two test cases, the complexity of the web site made it impossible for us to crawl with classical method for comparison, so we report the results separately in Section 4.3. Note that the number of test cases is not as large as we would like, but we are faced with the limitation of the tools we use to execute the crawl on RIAs[7]. We provide the characteristics of the model built for each of these nine RAIs in table 2. Note that these are the numbers for our component-based model, which is much smaller than the classical DOM-based model (see [27] for more details).

Table 2. Applications tested, along with their number of states and transitions in component-based model

Name	# States	# Trans.	Type	Name	# States	# Trans.	Type
Bebop	119	774	Simple	TestRIA	67	191	Test
Elfinder	152	3,239	Simple	Altoro	87	536	Test
Periodic	365	2,019	Simple	RTC	432	3,667	Complex
Clipmarks	31	377	Simple	MODX	1,291	7,868	Complex
DynaTable	24	49	Simple				

[5] `https://jazz.net/products/rational-team-concert`

[6] `http://modx.com`

[7] We stress that the work in question is not related to the strategy described here, but to the limitation of the available tools.

We have implemented all the mentioned crawling strategies in a prototype of IBM® Security AppScan® Enterprise[8]. Each strategy is implemented as a separate class in the same code base, so they use the same DOM equivalence mechanism [28], the same event identification mechanism [29], and the same embedded browser. For this reason, in Section 4.2 all strategies find the same model for each application. We crawl each application with a given strategy ten times and present the average of these crawls. In each crawl, the events of each state are randomly shuffled before they are passed to the strategy. The aim here is to eliminate influences that may be caused by exploring the events of a state in a given order since some strategies may explore the events on a given state sequentially.

4.2 Results on Simple RIAs

This first set of test case were simple enough to allow crawling with the traditional method. We report here comparisons with the greedy exploration [6] and the probability strategy [8], which are known to be to two most efficient strategy for building an exhaustive model [9]. This gives us complete knowledge of the model, allowing us to see whether our optimized strategy provides 100% coverage.

4.2.1 Complete Exploration Cost

Our first set of results are about the "total exploration costs", that is, the cost of finishing the crawl, expressed in terms of number of events executed. Most results are detailed in Figure 5. As can be seen, our component-based crawling method consistently outperforms both probability method and the greedy method by a very wide margin. The difference is more dramatic in RIAs that have a complex behavior, though even for the smaller ones, TestRIA and Altoro Mutual (not shown), the cost of component-based crawling is about 30% of the cost of the other methods. The best example among our test cases is Bebop, which contains very few data items shown on the page, but can sort and filter and expand/collapse those items in different manners. Even in an instance of the RIA with only 3 items, component-based crawling is 200 times more efficient than the other methods. This difference in performance quickly gets even bigger in an instance of the RIA with more items, as shown in Section 4.2.4.

4.2.2 Time Measurement

Since component-based crawling requires a fair amount of computation at each step, we also measured time in similar experiments to ensure this processing overhead does not degrade the overall performance. As can be seen on Table 3, even in terms of absolute time component-based crawling significantly outperforms the two other methods.

[8] Details are available at `http://ssrg.eecs.uottawa.ca/docs/prototype.pdf` Since our crawler is built on top of the architecture of a commercial product, we are not currently able to provide open-source implementations of the strategies.

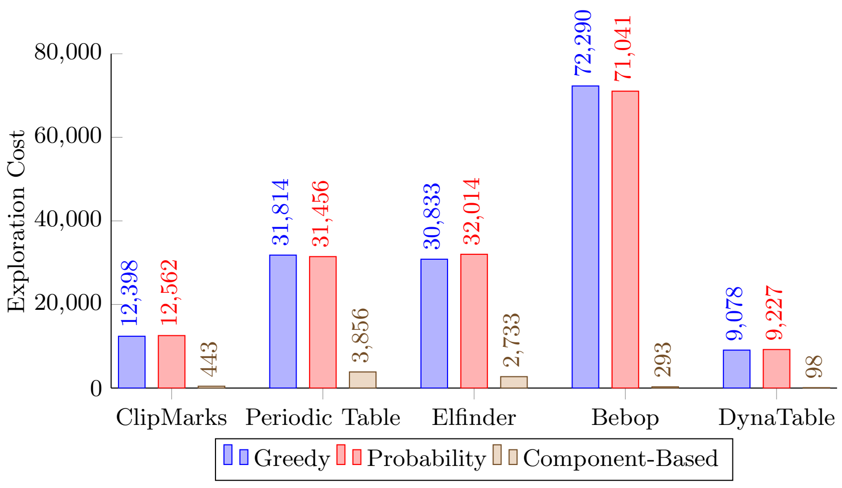


Fig. 5. Comparison of exploration costs of finishing crawl for different methods

Table 3. Time of finishing crawl for different methods (hh:mm:ss)

	TestRIA	Altor-Mutual	ClipMarks	Periodic Table	Elfinder	Bebop	DynaTable
Greedy	00 : 00 : 18	00 : 00 : 34	00 : 03 : 38	01 : 13 : 08	00 : 51 : 22	01 : 25 : 11	00 : 05 : 35
Probability	00 : 00 : 11	00 : 00 : 20	00 : 02 : 50	01 : 09 : 42	00 : 49 : 00	01 : 17 : 32	00 : 04 : 51
Component-Based	00 : 00 : 06	00 : 00 : 04	00 : 00 : 13	00 : 01 : 21	00 : 08 : 21	00 : 00 : 29	00 : 00 : 06

4.2.3 Coverage

Unlike previous DOM-based methods, component-based crawling does not guaranty complete coverage. This is because the method is based on discovering automatically independent components, and if the method wrongly identifies as "components" sections of the DOM that are not independent from each other, some coverage might be lost. It is difficult to know in general the amount of coverage that can be lost, but in the case of our seven test cases, our method systematically reached 100% coverage. No information was lost despite the dramatic decrease in crawling time.

4.2.4 Scalability

Some of the test RIAs that we used had to be significantly "trimmed" before they could be crawled by the traditional methods. One example is Clipmark, which displays a number of items on the page. Although it had initially 40 items, we had to reduce it to 3 in order to finish the crawl with the traditional methods! When using component-based crawling, on all of our test beds we were easily able to finish the crawl on the original data, and we could increase the number of items beyond that without problem. We show the data for two examples, Clipmarks on Figure 6 and Bebop on Figure 7. As can clearly been seen on both examples, while the crawling time increases linearly with the number of items in the page when using component-based crawling, it grows exponentially with the DOM-based, greedy method and soon becomes intractable. The results are the same with probability, and will *necessarily* be similar for *any* DOM-based crawling method, since the size of the end-model itself increases exponentially with the number of items in the page. This shows that component-based crawling is able to crawl and index RIAs that are simply out of reach to any DOM-based strategy.

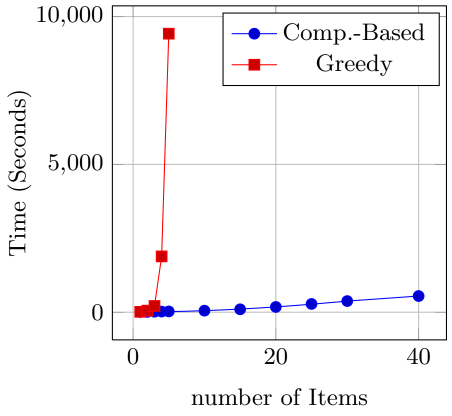


Fig. 6. Scalability with Clipmarks

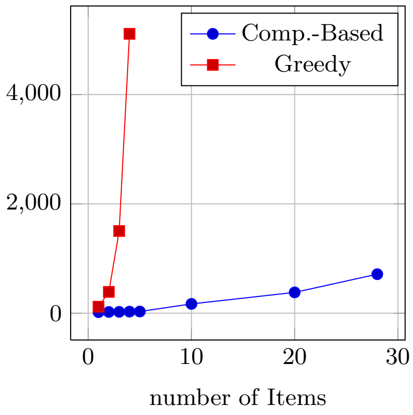


Fig. 7. Scalability with BeBop

4.3 Results on Complex RIAs

When crawling large and complex RIAs, comparison with DOM-based methods do not tell much, since these methods are essentially unable to crawl them. In addition, for many of these RIAs (e.g. Facebook, Gmail etc.), the amount of data available is very large. "Finishing" the crawl is often not a realistic proposition. Instead, the question becomes how efficient the crawl can be as it progresses overtime. In order to measure this, we focus on the question of crawling for indexing, where we argue that a fair definition of "efficient" is the ability of the crawler to keep finding new content. In our experiment, we have measured how much the textual information accumulated increases overtime. An efficient method will provide a steady increasing amount of information, while an inefficient method might stop providing any new information for long period of times (basically re-fetching known data many times). We have measured how "efficient" component-based crawling is, by counting how many "lines" of text (excluding html tags) are accumulated overtime. As can been seen from the Figures 8 and 9, for both of our examples, the method provides a nice steadily increasing line, showing that the method is efficient at fetching new information overtime. An inefficient method would have plateaued, during which the crawl is not adding any new data.

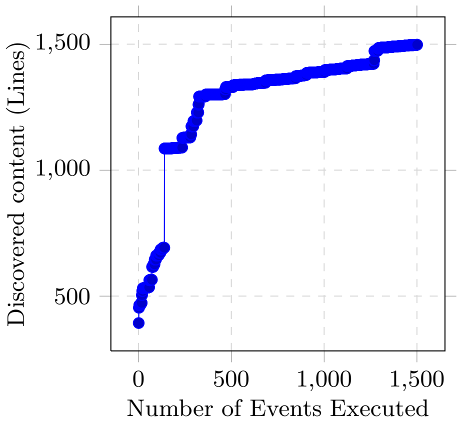


Fig. 8. Progress overtime with RTC

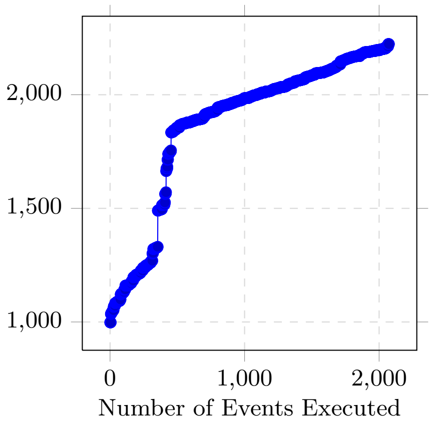


Fig. 9. Progress overtime with MODX

5 Conclusions and Future Work

This work addresses one of the most prevalent problems in the context of crawling AJAX-based RIAs: state space explosion. The presented method detects independent components and models the RIA at component level rather than DOM level, resulting in exponential reduction of the overall crawling complexity, with minimal or no loss of data coverage. The method captures the effect of event execution more precisely, resulting in a simpler model with fewer states and transitions. The produced model can point to any desired data with an event execution trace from the initial state, but cannot necessarily produce a path to lead to any valid DOM-state. The methods has been implemented using a greedy exploration strategy and DOM diff as automatic component discovery algorithm. Our experimental results verify the significant performance gain of the method while covering equal content as DOM-based methods.

This work can be improved in several areas. In particular, our future work include devising better algorithms for component discovery. One important improvement can be done on detection and handling of violations: currently, we have no effective way of recovering from a situation where components that have been assumed to be independent turn out not to be. We simply ignore theses violation. Although that did not impact negatively our experimental results, we cannot be sure that this won't be the case on every RIAs. A naive approach for detecting violations and adapting the strategy accordingly is not particularly difficult, but it would be too costly to be practical.

Acknowledgments. This work is partially supported by the IBM Center for Advanced Studies, and the Natural Sciences and Engineering Research Council of Canada (NSERC). The views expressed in this article are the sole responsibility of the authors and do not necessarily reflect those of IBM.

References

1. Fraternali, P., Rossi, G., Sánchez-Figueroa, F.: Rich internet applications. IEEE Internet Computing 14(3), 9–12 (2010)
2. Duda, C., Frey, G., Kossmann, D., Zhou, C.: Ajaxsearch: crawling, indexing and searching web 2.0 applications. Proceedings of the VLDB Endowment 1(2), 1440–1443 (2008)
3. Duda, C., Frey, G., Kossmann, D., Matter, R., Zhou, C.: Ajax crawl: making ajax applications searchable. In: ICDE 2009, pp. 78–89. IEEE (2009)
4. Amalfitano, D., Fasolino, A.R., Tramontana, P.: Reverse engineering finite state machines from rich internet applications. In: Proceedings of WCRE, pp. 69–73. IEEE (2008)
5. Amalfitano, D., Fasolino, A.R., Tramontana, P.: Rich internet application testing using execution trace data. In: Proceedings of ICSTW, pp. 274–283. IEEE (2010)
6. Peng, Z., He, N., Jiang, C., Li, Z., Xu, L., Li, Y., Ren, Y.: Graph-based ajax crawl: Mining data from rich internet applications. In: Proceedings of ICCSEE, vol. 3, pp. 590–594 (March 2012)
7. Dincturk, M.E., Jourdan, G.V., Bochmann, G.v., Onut, I.V.: A model-based approach for crawling rich internet applications. ACM Transactions on the WEB (to appear, 2014)
8. Choudhary, S., Dincturk, M.E., Mirtaheri, S.M., Jourdan, G.-V., Bochmann, G.v., Onut, I.V.: Model-based rich internet applications crawling:menu and probability models. Journal of Web Engineering 13(3) (to appear, 2014)
9. Choudhary, S., Dincturk, M.E., Mirtaheri, S.M., Jourdan, G.-V., Bochmann, G.v., Onut, I.V.: Building rich internet applications models: Example of a better strategy. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 291–305. Springer, Heidelberg (2013)
10. Faheem, M., Senellart, P.: Intelligent and adaptive crawling of web applications for web archiving. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 306–322. Springer, Heidelberg (2013)
11. Amalfitano, D., Fasolino, A.R., Polcaro, A., Tramontana, P.: The dynaria tool for the comprehension of ajax web applications by dynamic analysis. In: Innovations in Systems and Software Engineering, pp. 1–17 (2013)
12. Doush, I.A., Alkhateeb, F., Maghayreh, E.A., Al-Betar, M.A.: The design of ria accessibility evaluation tool. Advances in Engineering Software 57, 1–7 (2013)
13. Mesbah, A., van Deursen, A.: Invariant-based automatic testing of ajax user interfaces. In: ICSE, pp. 210–220 (May 2009)
14. Amalfitano, D., Fasolino, A.R., Tramontana, P.: A gui crawling-based technique for android mobile application testing. In: Proceedings of ICSTW, pp. 252–261. IEEE Computer Society, Washington, DC (2011)

15. Amalfitano, D., Fasolino, A.R., Tramontana, P., De Carmine, S., Memon, A.M.: Using gui ripping for automated testing of android applications. In: Proceedings of ASE, pp. 258–261. ACM, New York (2012)
16. Erfani, M., Mesbah, A.: Reverse engineering ios mobile applications. In: Proceedings of WCRE (2012)
17. Mesbah, A., Bozdag, E., van Deursen, A.: Crawling ajax by inferring user interface state changes. In: Proceedings of ICWE, pp. 122–134. IEEE (2008)
18. Ayoub, K., Aly, H., Walsh, J.: Dom based page uniqueness identification, canada patent ca2706743a1 (2010)
19. Milani Fard, A., Mesbah, A.: Feedback-directed exploration of web applications to derive test models. In: Proceedings of ISSRE, 10 pages. IEEE Computer Society (2013)
20. Choudhary, S., Dincturk, M.E., Mirtaheri, S.M., Moosavi, A., Bochmann, G.v., Jourdan, G.-V., Onut, I.-V.: Crawling rich internet applications: the state of the art. In: CASCON, pp. 146–160 (2012)
21. Mirtaheri, S.M., Dinçtürk, M.E., Hooshmand, S., Bochmann, G.v., Jourdan, G.-V., Onut, I.V.: A brief history of web crawlers. In: Proceedings of CASCON, pp. 40–54. IBM Corp. (2013)
22. Bezemer, C.P., Mesbah, A., van Deursen, A.: Automated security testing of web widget interactions. In: Proceedings of ESEC/FSE, pp. 81–90. ACM (2009)
23. Chen, A.Q.: Widget identification and modification for web 2.0 access technologies (wimwat). ACM SIGACCESS Accessibility and Computing (96), 11–18 (2010)
24. Crescenzi, V., Mecca, G., Paolo, Merialdo, et al.: Roadrunner: Towards automatic data extraction from large web sites. In: VLDB, vol. 1, pp. 109–118 (2001)
25. Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming 8(3), 231–274 (1987)
26. Peng, Z., He, N., Jiang, C., Li, Z., Xu, L., Li, Y., Ren, Y.: Graph-based ajax crawl: Mining data from rich internet applications. In: Proceedings of ICCSEE, vol. 3, pp. 590–594. IEEE (2012)
27. Moosavi, A.: Component-based crawling of complex rich internet applications. Master's thesis, EECS - University of Ottawa (2014), `http://ssrg.site.uottawa.ca/docs/Ali-Moosavi-Thesis.pdf`
28. Benjamin, K., Bochmann, G.v., Jourdan, G.-V., Onut, I.-V.: Some modeling challenges when testing rich internet applications for security. In: Proceedings of ICSTW, pp. 403–409. IEEE Computer Society, Washington, DC (2010)
29. Choudhary, S., Dincturk, M.E., Bochmann, G.v., Jourdan, G.-V., Onut, I.V., Ionescu, P.: Solving some modeling challenges when testing rich internet applications for security. In: Proceedings of ICST, pp. 850–857 (2012)

Pattern-Based Specification of Crowdsourcing Applications

Alessandro Bozzon[1], Marco Brambilla[2], Stefano Ceri[2], Andrea Mauri[2], and Riccardo Volonterio[2]

[1] Software and Computer Technologies Department, Delft University of Technology, Postbus 5 2600 AA, Delft, The Netherlands
a.bozzon@tudelft.nl

[2] Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB) Politecnico di Milano, Piazza Leonardo da Vinci, 32, 20133 Milano, Italy
{name.surmame}@polimi.it

Abstract. In many crowd-based applications, the interaction with performers is decomposed in several tasks that, collectively, produce the desired results. Tasks interactions give rise to arbitrarily complex workflows. In this paper we propose methods and tools for designing crowd-based workflows as interacting tasks. We describe the modelling concepts that are useful in such framework, including typical workflow patterns, whose function is to decompose a cognitively complex task into simple interacting tasks so that the complex task is co-operatively solved.

We then discuss how workflows and patterns are managed by Crowd-Searcher, a system for designing, deploying and monitoring applications on top of crowd-based systems, including social networks and crowd-sourcing platforms. Tasks performed by humans consist of simple operations which apply to homogeneous objects; the complexity of aggregating and interpreting task results is embodied within the framework. We show our approach at work on a validation scenario and we report quantitative findings, which highlight the effect of workflow design on the final results.

1 Introduction

Crowd-based applications are becoming more and more widespread; their common aspect is that they deal with solving a problem by involving a vast set of performers, who are typically extracted from a wide population (the "crowd"). In many cases, the problem is expressed in the form of simple questions, and the performers provide a set of answers; a software system is in charge of organising a crowd-based computation – typically by distributing questions, collecting responses and feedbacks, and organising them as a well-structured result of the original problem.

Crowdsourcing systems, such as Amazon Mechanical Turk (AMT), are natural environments for deploying such applications, since they support the assignment to humans of simple and repeated tasks, such as translation, proofing, content tagging and items classification, by combining human contribution and automatic analysis of results [1]. But a recent trend (emerging, e.g., during the CrowdDB Workshop[1]), is to use many other kinds of platforms for engaging crowds, such as proprietary community-building systems (e.g., FourSquare or Yelp) or general-purpose social networks (e.g., Facebook or Twitter). In the various platforms, crowds take part to social computations both for monetary rewards and for non-monetary motivations, such as public recognition, fun, or genuine will of sharing knowledge.

In previous work, we presented CrowdSearcher [2, 3], offering a conceptual framework, a specification paradigm and a reactive execution control environment for designing, deploying, and monitoring applications on top of crowd-based systems, including social networks and crowdsourcing platforms. In Crowdsearcher, we advocate a top-down approach to application design which is independent on the particular crowd-based system. We adopt an abstract model of crowdsourcing activities in terms of elementary task types (such as: labelling, liking, sorting, classifying, grouping) performed upon a data set, and then we define a crowdsourcing task as an arbitrary composition of these task types; this model does not introduce limitations, as arbitrary crowdsourcing tasks can always be defined by aggregating several operation types or by decomposing the tasks into smaller granularity tasks, each one of the suitable elementary type. In general, an application cannot be submitted to the crowd in its initial formulation; transformations are required to organise and simplify the initial problem, by structuring it into a *workflow of crowd-based tasks* that can be effectively performed by individuals, and can be submitted and executed, possibly in parallel. Several works [4, 5] have analysed typical *crowdsourcing patterns*, i.e. typical cooperative schemes used for organising crowd-based applications.

The goal of this paper is to present a systematic approach to the design and deployment of crowd-based applications as arbitrarily complex workflows of elementary tasks, which emphasises the use of crowdsourcing patterns. While our previous work was addressing the design and deployment of a single task, in this paper we model and deploy applications consisting of arbitrarily complex task interactions, organised as a workflow; we use either *data streams* or *data batches* for data exchange between tasks, and illustrate that tasks can be controlled through *tight coupling* or *loose coupling*. We also show that our model supports the known crowd management patterns, and in particular we use our model as a unifying framework for a systematic classification of patterns.

The paper is structured as follows. Section 2 presents related work; Section 3 introduces the task and workflow models and design processes. Section 4 details a set of relevant crowdsourcing patterns. Section 5 illustrates how workflow specifications are embodied within the execution control structures of Crowdsearcher, and finally Section 6.3 discusses several experiments, showing how differences in workflow design lead to different application results.

[1] `http://dbweb.enst.fr/events/dbcrowd2013/`

2 Related Work

Many crowdsourcing startups[2] and systems [6] have been proposed in the last years. Crowd programming approaches rely on imperative programming models to specify the interaction with crowdsourcing services (e.g., see *Turkit* [7], *RABJ* [8], *Jabberwocky* [9]). Several programmatic methods for human computation have been proposed [7][8][9][10], but they do not support yet the complexity required by real-world, enterprise–scale applications, especially in terms of designing and controlling complex flows of crowd activities.

Due to its flexibility and extensibility, our approach covers the expressive power exhibited by any of the cited systems, and provides fine grained targeting to desired application behaviour, performer profiles, and adaptive control over the executions.

Several works studied how to involve humans in the creation and execution of workflows, and how to codify common into modular and reusable patterns. Process-centric workflow languages [11] define business artefacts, their transformations, and interdependencies trough tasks and their dependencies. Scientists and practitioners put a lot of effort in defining a rich set of control-driven workflow patterns.[3] However, this class of process specification languages: focus mainly on control flow, often abstracting away data almost entirely; disregard the functional and non-functional properties of the involved resources; do not specify intra- and inter-task execution and performer controls; and provide no explicit modelling primitives for data processing operations.

In contrast, data- driven workflows have recently become very popular, typically in domains where database are central to processes [12][13], and data consistency and soundness is a strong requirement. Data-driven workflows treat data as first-class citizens, emphasise the role of control intra- and inter-task control, and ultimately served as an inspiration for our work.

Very few works studied workflow-driven approaches for crowd work. CrowdLang [5] is notable exception, which supports process-driven workflow design and execution of tasks involving human activities, and provides an executable model-based programming language for crowd and machines. The language, however, focuses on the modelling of coordination mechanisms and group decision processes, and it is oblivious to the design and specification of task-specific aspects.

Several works tried to codify patterns for crowdsourcing. At task level, a great wealth of approaches has been proposed for the problems of output agreement [14], and performer control [15]. At workflow level, less variety can be witnessed, but a set of very consolidated patterns emerge [7][16][4][17][18]. In Section 4 we will provide an extensive review of the most adopted pattern in crowdsourcing, classifying them in the light of the workflow model of Section 3.

[2] E.g., CrowdFlower, Microtask, uTest.

[3] `http://workflowpatterns.com/`

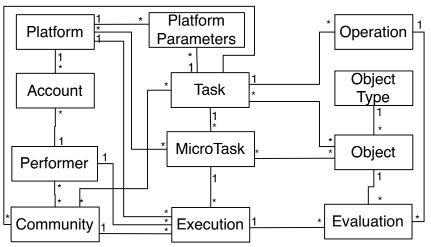


Fig. 1. Metamodel of task properties

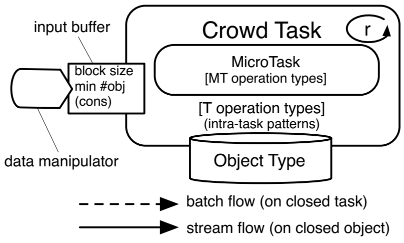


Fig. 2. Task notation

3 Models and Design of Crowd-Based Workflows

Although humans are capable of solving complex tasks by using their full cognitive capacity, the approaches used in crowd-based computations prefer to decompose complex tasks into simpler tasks and then elaborate their results [16]. Following this approach, we restrict crowdsourcing tasks be to simple operations which apply to homogeneous objects; operations are simple actions (e.g. labelling, liking, sorting, classifying, grouping, adding), while objects have an arbitrary schema and are assumed to be either available to the application or to be produced as effect of application execution.

3.1 Task Model

Tasks of a crowd-based application are described in terms of an abstract model, that was initially presented in [2], and represented in Fig. 1. We assume that each **task** receives as input a list of **objects** (e.g., photos, texts, but also arbitrarily complex objects, all conforming to the same **object type**) and asks performers to do one or more **operations** upon them, which belong to a predefined set of abstract **operation types**. Examples of operation types are *Like*, for assigning a preference to an item; or *Classify*, for assigning each item to one or more classes. The full list of currently supported operation types is reported in [2]. Task management requires specific sets of objects to be assembled into a unit of execution, called **micro-task**, that is associated with a given **performer**. Each micro-task can be invited or executed on different **platforms** and/or **communities**. The relation with platform is specified through a series of **platform parameters**, specific for each platform, that are needed in order retrieve the answers of the performers (e.g., the HIT identifier on AMT). A **performer** may be registered on several platforms (with different accounts) and can be part of several communities. Micro-task **execution** contains some statistics (e.g., start and end timestamps). The **evaluation** contains the answer of the performer for each object, whose schema depends on the operation type.

For example, a *like* evaluation is a counter that registers how many performers like the object, while a *classify* evaluation contains the category selected by the performers for that object.

3.2 Workflow Model

A **crowdsourcing workflow** is defined as a control structure involving two or more interacting tasks performed by humans. Tasks have an input buffer that collects incoming data objects, described by two parameters: 1) The **task size**, i.e. the minimum number of objects (m) that allow starting a task execution; 2) The **block size**, i.e. the number of objects (n) consumed by each executions.

Clearly, $n \leq m$, but in certain cases at least m objects must be present in the buffer before starting an execution; in fact n can vary between 1 and the whole buffer, when a task execution consumes all the items currently in the buffer. Task execution can cause **object removal**, when objects are removed from the buffer, or **object conservation**, when objects are left in the buffer, and in such case the term **new items** denotes those items loaded in the buffer since the last execution.

Tasks communicate with each other with **data flows**, produced by extracting objects from existing data sources or by other tasks, as streams or batches. **Data streams** occur when objects are communicated between tasks one by one, typically in response to events which identify the completion of object's computations. **Data batches** occur when all the objects are communicated together from one task to another, typically in response to events related to the closing of task's computations.

Flows can be constrained based on a condition associated with the arrow representing the flow. The condition applies to properties of the produced objects and allows transferring only the instances that satisfy the condition. Prior to task execution, a **data manipulator** may be used to compose the objects in input to a task, possibly by merging or joining incoming data flows.

We can represent tasks within workflows as described in Fig. 2, where each task is equipped with an input buffer and an optional data manipulator, and may receive data streams or data batches from other tasks. Each task consists of micro-tasks which perform given operations upon objects of a given object type; the parameter r indicates the number of executions that are performed for each micro-tasks, when statically defined (default value is 1). Execution of tasks can be performed according to intra-task patterns, as described in Section 4.

3.3 Workflow Design

Workflow design consists of designing tasks interaction; specifically, it consists of defining the workflow schema as a directed graph whose nodes are tasks and whose edges describe dataflows between tasks, distinguishing streams and batches. In addition, the coupling between tasks working on the same object type can be defined as loose or tight.

Loose coupling is recommended when two tasks act independently upon the objects (e.g. in sequence); although it is possible that the result of one task may have side effects on the other task, such side effects normally occur as an exception and affect only a subset of the objects. Loosely coupled tasks have independent control marts and monitoring rules (as described in Section 3.4).

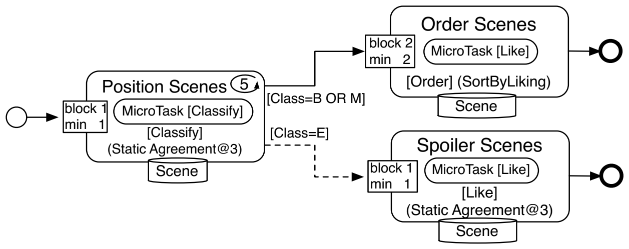


Fig. 3. Example of crowd flow

Tight coupling is recommended when the tasks intertwine operations upon the same objects, whose evolution occurs as combined effect of the tasks' evolution; tightly coupled tasks share the same control mart and monitoring rules.

Figure 3 shows a simple workflow example in the domain of movie scenes annotation. The *Position Scenes* tasks asks performers to say whether a scene appears at the beginning, middle or end of the film; it is a classification task, one scene at a time, with 5 repetitions and acceptance of results based on an agreement threshold of 3. Scenes in the ending part of the movies are transmitted to the *Spoiler Scenes* task, which asks performers whether the scene is a spoiler or not;[4] scenes at the beginning or in the middle of the movie are transmitted to the *Order Scenes* task, which asks performers to order them according to the movie script; each micro-task orders just two scenes, by asking the performer to select the one that comes first. The global order is then reconstructed. Given that all scenes are communicated within the three tasks, they are considered as tightly coupled.

3.4 Task Design

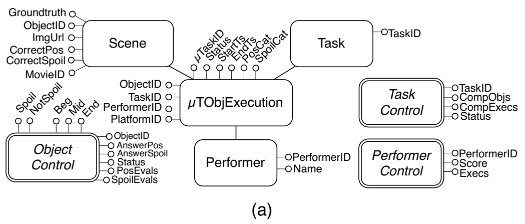
Crowdsourcing tasks are targeted to a single object type and are used in order to perform simple operations which either apply to a single object (such as **like**, **tag**, or **classify**) or require comparison between objects (such **choice** or **score**); more complex tasks perform operations inspired by database languages, such as **select**, **join**, **sort**, **rank**, or **group by**.

Task design consists of the following phases: 1) **Operations design** – deciding how a task is assembled as a set of operation types; 2) **Object and performer design** – defining the set of objects and performers for the task; 3) **Strategy design** – Defining how a task is split into micro-tasks, and how micro-tasks are assigned to subsets of objects and performers; 4) **Control Design** – Defining the rules that enable the run-time control of objects, tasks, and performers.

For monitoring task execution, a data structure called **control mart** was introduced in [3]; Control consists of four aspects:

- **Object control** is concerned with deciding when and how responses should be generated for each object.

[4] A *spoiler* is a scene that gives information about the movie's plot and as such should not be used in its advertisement.



```
RULE: CountBeginning
e:  UPDATE FOR Execution[Answer]
c:  NEW.Answer == 'Beginning'
a:  SET Object_CTRL[oid == NEW.oid].Beg += 1
```

(b)

Fig. 4. (a) Example of control mart for the tasks of Fig. 3; (b) Example of control rule that updates the number of responses in the *Position of Scenes* task

- **Performer control** is concerned with deciding how performers should be dynamically selected or rejected, on the basis of their performance.
- **Task control** is concerned with completing a task or re-planning task execution.

The control of objects, performers and tasks is performed by **active rules**, expressed according to the *event-condition-action* (ECA) paradigm. Each rule is triggered by **events** (e) generated upon changes in the control mart or periodically; the rule's **condition** (c) is a predicate that must be satisfied on order for the action to be executed; the rule's **actions** (a) change the content of the control mart. Rules properties (e.g., termination) can be been proven in the context of a well-organised computational framework [3].

Figure 4(a) shows a sample control mart for the three tasks in the example scenario, which we assume to be tightly connected, thus using the same data mart. The control mart stores all the required information for controlling the task's evolution and is automatically defined from the task specifications. Figure 4(b) reports a simple control rule that updates the number of responses with value "Beginning" after receiving an answer.
This rule has the following behaviour: every time a performer perform a new evaluation on a specific object (UPDATE event on μTObjExecution), if the selected answer is "Beginning" (the condition part of the rule), then it increases the counter of the "Beginning" category for that object (Object_CTRL[oid == New.oid] selected the correct object, then the correct property can be acessed with the dot notation). For a deeper description of the rule grammar and structure see our previous work [3].

4 Crowdsourcing Patterns

Several patterns for crowd-based operations are defined in the literature. We review them in light of the workflow model of Section 3. We distinguish them in three classes and we implement them in Crowdsearcher (see Section 5):

- **Intra-Task Patterns.** They are typically used for executing a complex task by means of a collection of operations which are cognitively simpler than the original task. Although these patterns do not appear explicitly in the workflow, they are an essential ingredient of crowd-based computations.
- **Workflow Patterns.** They are used for solving a problem by involving different tasks, which require a different cognitive approach; results of the different tasks, once collected and elaborated, solve the original problem.
- **Auxiliary Patterns.** They are typically performed before or after both intra-task and workflow patterns in order either to simplify their operations or to improve theirs results.

4.1 Intra-Task Patterns

Intra-task patterns apply to complex operations, whose result is obtained by composing the results of simpler operations. They focus on problems related to the planning, assignment, and aggregation of micro tasks; they also include quality and performer control aspects. Figure 5 describes the typical set of design dimensions involved in the specification of a task. When the operation applies to a large number of objects and as such cannot be mapped to a single pattern instantiation, it is customary to put in place a *splitting strategy*, in order to distribute the work, followed by an *aggregation strategy*, to put together results. This is the case in many data-driven tasks stemming from traditional relational data processing which are next reviewed.

Consensus Patterns. The most commonly used intra-task patterns aim at producing responses by replicating the operations which apply to each object, collecting multiple assessments from human workers, and then returning the answer which is more likely to be correct. These patterns are referred to as *consensus* or *agreement patterns*. Typical consensus patterns are: *a*) **StaticAgreement** [3]: accepts a response when it is supported by a given number of performers. For instance, in a tag operation we consider as valid responses all the tags that have been added by at least 5 performers. *b*) **MajorityVoting** [19]: accepts a response only if a given number of performers produce the same response, given a fixed number of total executions. *c*) **ExpectationMaximisation** [20]: adaptively alternates between estimating correct answers from task parameters (e.g. complexity), and estimating task parameters from the estimated answers, eventually converging to maximum-likelihood answer values.

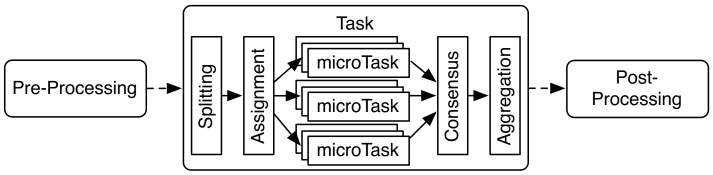


Fig. 5. Building blocks of an Intra-Task Pattern

Join Patterns. Crowd join patterns, studied in [14], are used to build an equality relationship between matching objects in the context of crowdsourcing tasks. We identify: *a*) **SimpleJoin** consists in defining microtasks performing a simple classification operation, where each execution contains a single pair of items to be joined, together with the join predicate question, and two buttons (Yes, No) for responding whether the predicate evaluates to true or false; *b*) **OneToManyJoin** is a simple variant that includes in the same microtask one left object and several right candidates to be joined; *c*) **ManyToManyJoin** includes in the same microtask several candidate pairs to be joined;

Sort Patterns. Sort patterns determine the total ordering of a set of input objects. The list includes: *a*) **SortByGrouping** [14] orders a large set of objects by aggregating the results of the ordering of several small subsets of them. *b*) **SortByScoring** [14] asks performers to rate each item in the dataset according to a numerical scale. *c*) **SortByLiking** [3] is a variant that simply asks the performer to select/like the items they prefer. The mean (or sum) of the scores achieved by each image is used to order the dataset. *d*) **SortByPairElection** [3] asks workers to perform a pairwise comparison of two items and indicate which one they like most. Then ranking algorithms calculate their ordering. *e*) **SortByTournament** [18], presents to performers a tournament-like structure of sort tasks; each tournament elect its champions that progress to the next level, eventually converging to a final order.

Grouping Patterns. Grouping patterns are used in order to classify or clustered several objects according to their properties. We distinguish:
a) **GroupingByPredefinedClasses**[21] occurs when workers are provided with a set of known classes. *b*) **GroupingByPreference** [22] occurs when groups are formed by performers, for instance by asking workers to select the items they prefer the most, and then clustering inputs according to ranges of preferences.

Performer Control Patterns. Quality control of performers consists in deciding how to engage qualified workers for a given task and how to detect malicious or poorly performing workers. The most established patterns for performer control include: *a*) **QualificationQuestion** [23], at the beginning of a microtask, for assessing the workers expertise and deciding whether to accept his contribution or not. *b*) **GoldStandard**, [3] for both training and assessing worker's quality through a initial subtask whose answers are known (they belong to the so-called *gold truth*. *c*) **MajorityComparison**, [3] for assessing performers' quality against responses of the majority of other performers, when no gold truth is available.

4.2 Auxiliary Intra-Task Patterns

The above tasks can be assisted by auxiliary operations, performed before or after their executions, as shown in Figure 5. *Pre-processing steps* are in charge of assembling, re-shaping, or filtering the input data so to ease or optimise the main task. *Post-processing steps* is typically devoted to the refinement or transformation of the task outputs into their final form.

Examples of auxiliary patterns are: *a*) **PruningPattern** [14], consisting of applying simple preconditions on input data in order to reduce the number of evaluations to be performed. For instance, in a join task between sets of actors (where we want to identify the same person in two sets), classifying items by gender, so as to compared only pairs of the same gender. *b*) **TieBreakPattern** [14], used when a sorting task produces uncertain rankings (e.g. because of ties in the evaluated item scores); the post-processing includes an additional step that asks for an explicit comparison of the uncertainly ordered items.

4.3 Workflow Patterns

Very often, a single type of task does not suffice to attain the desired crowd business logic. For instance, with open-ended multimedia content creation and/or modification, it is difficult to assess the quality of a given answer, or to aggregate the output of several executions. A **Workflow Pattern** is a workflow of heterogeneous crowdsourcing tasks with co-ordinated goals. Several workflow patterns defined in the literature are next reviewed; they are comparatively shown in Figure 6:

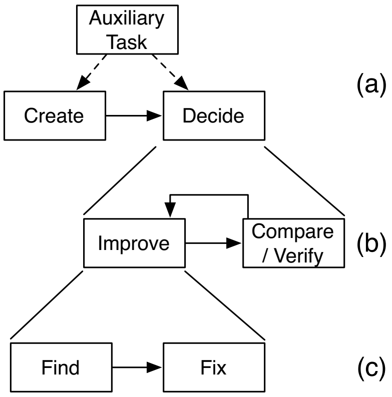


Fig. 6. Template for complex task patterns

a) **Create/Decide** [16], shown in Figure 6(a), is a two-staged pattern where first workers create various options for new content, then a second group of workers vote for the best option. Note that the *create* step can include any type of basic task. This pattern can have several variants: for instance, with a stream data flow, the vote is typically restricted to the solutions which are produced faster, while with a batch data flow the second task operates on all the generated content, in order to pick the best option overall. *b*) **Improve/Compare** [7], shown in Figure 6(b), iterates on the decide step to progressively improve the result. In this pattern, a first pool of workers creates a first version of a content; upon this version, a second pool of workers creates an improved version, which is then compared, in a third task, to decide which one is the best (the original or the improved one). The improvement/compare cycle can be repeated until the improved solution is deemed as final. *c*) **Find/Fix/Verify** [4], shown in Figure 6(c), further decomposes the *improve* step, by splitting the task of finding potential improvements from the task of actually implementing them.

4.4 Auxiliary Workflow Patterns

Auxiliary tasks can be designed to support the creation and/or the decision tasks. They include: *a*) **AnswerBySuggestion** [17]: given a create operations as input, the provided solution can be achieved by asking suggestions from the crowd as follows. During each execution, a worker can choose one of two actions: it can either stop and submit the most likely answer, or it can create another job and receive another response to the task from another performer. The auxiliary suggestion task produces content that can be used by the original worker to complete or improve her answer. *b*) **ReviewSpotcheck** strengthens the decision step by means of a two-staged review process: an additional quality check is performed after the corrections and suggestions provided by the performers of the decision step. The revision step can be performed by the same performer of the decision step or by a different performer.

5 Workflow Execution

Starting from the declarative specification described in Sections 3 and 4, an automatic process generates task descriptors and their relations. Single tasks and their internal strategies and patterns are transformed into executable specification; we support all the intra-task patterns described in Section 4, through model transformations that generate the control marts and control rules for each task [3]. Task interactions are implemented differently depending on whether interacting tasks are tightly coupled or loosely coupled.

Tightly coupled tasks share the control mart structure (and the respective data instances), thus coordination is implemented directly on data. Each task posts its own results and control values in the mart. Dependencies between tasks are transformed into rules that trigger the creation of new micro-tasks and their executions, upon production of new results by events of object or task closure.

Loosely coupled tasks have independent control marts, hence their interaction is more complex. Each task produces in output events such as `ClosedTask`, `ClosedObject`, `ClosedMicrotask`, `ClosedExecution`. We rely on an event based, publish-subscribe mechanism, which allows tasks to be notified by other tasks about some happening. Loosely coupled tasks do not rely on a shared data space, therefore events carry with them all the relevant associated pieces of information (e.g., a `ClosedObject` event carries the information about that object; a `ClosedTask` event carries the information about the closed objects of the task).

The workflow structure dictates how tasks subscribe to events of other tasks. Once a task is notified by an incoming event, the corresponding data is incorporated in its control mart by a-priori application of the data manipulation program, specified in the data manipulator stage of the task. Then, reactive processing takes place within the control mart of the task.

Modularity allows executability through model transformations which are separately applied to each task specification. Automatically generated rules and mart structures can be manually refined or enriched when non-standard behaviour is needed.

This approach is supported by CrowdSearcher, a platform for crowd management written in JavaScript. CrowdSearcher runs on *Node.js*, a full-fledged event-based system, which fits the need of our rule-based approach. Each control rule is translated into scripts; triggering is modelled through internal platform

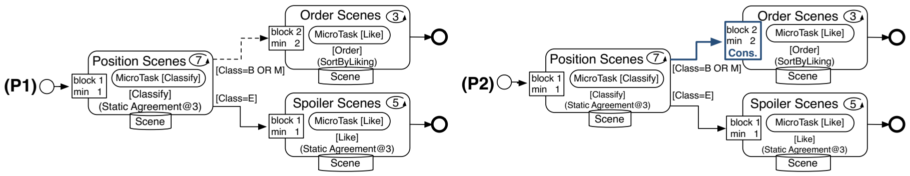


Fig. 7. Flow variants for the Positioning scenario

events. Precedence between rules is implicitly obtained by defining the scripts in the proper order. CrowdSearcher offers a cloud-based environment to transparently interface with social networks and crowdsourcing platforms, according to the task model described in Section 3.1. It features an online configuration interface where designers build complex crowdsourcing applications through a wizard–driven, step by step approach. A built-in *Task Execution Framework* (TEF) provides support for the creation of custom task user interfaces, to be deployed as stand-alone application, or embedded within third-party platforms such as Amazon Mechanical Turk. Specific modules are devoted to the invitation, identification, and management of performers, thus offering support for a broad range of expert selection paradigms, from pure pull approaches of open marketplaces, to pre-assigned execution to selected performers. Alternatives for the implementation of operations on crowd-based systems are discussed in [2].

6 Experiments

We demonstrate various pattern-based workflow scenarios, defined using our model and method and deployed by using Crowdsearcher as design framework and Amazon Mechanical Turk as execution platform. We consider several scenes taken from popular movies, and we enrich them with crowd-sourced information regarding their position in the movie, whether the scene is a spoiler, and the presence of given actors in each scene. In the experiments reported here we considered the movie "The Lord of the Rings: the Fellowship of the Ring". We extracted 20 scenes and we created a groundtruth dataset regarding temporal positioning and actors playing in the scenes. We compare cost and quality of executions for different workflow configurations.

Table 1. *Scenario 1 (Positioning)*: number of evaluated objects, microtask executions, elapsed execution time, performers, and executions per performer (for each task and for each scenario configuration)

	Position Scenes (payed $0.01)					**Order Scene (payed $0.01)**					**TOTAL**		
	#Obj	***#Exe***	***Time***	***#Perf***	***#Exe/Perf***	***#Obj***	***#Exe***	***Time***	***#Perf***	***#Exe/Perf***	**Time**	**Cost**	**#Perf**
P1	20	147	123	16	9.19	17	252	157	14	18.00	342	3.99$	26
P2	20	152	182	12	12.67	17	230	318	17	13.53	349	3.82$	26

Table 2. *Scenario 2 (Actor)*: number of evaluated objects, microtask executions, elapsed execution time, performers, and executions per performer (for each task and for each scenario configuration)

	Find Actors (payed $0.03)					**Validate Actors (payed $0.02)**					**TOTAL**		
	#Obj	*#Exe*	*Time*	*#Perf*	*#Exe/Perf*	*#Obj*	*#Exe*	*Time*	*#Perf*	*#Exe/Perf*	**Time**	**Cost**	**#Perf**
A1	20	100	120	18	5.56	–	–	–	–	–	120	3.00$	18
A2	20	100	128	10	10.00	–	–	–	–	–	128	3.00$	10
A3	20	100	123	14	7.15	20	21	154	10	2.10	159	3.42$	20
A4	20	100	132	10	10.00	41	19	157	9	2.10	164	3.38$	16
A5	20	100	126	13	7.69	69	60	242	17	3.53	257	4.20$	24
A6	66	336	778	56	6.00	311	201	821	50	4.02	855	14.10$	84

6.1 Scenario 1: Scene Positioning

The first scenario deals with extracting information about the temporal position of scenes in the movie and whether they can be considered as as spoilers. Two variants of the scenario have been tested, as shown in Figure 7: the task *Position Scenes* classifies each scene as belonging to the beginning, middle or ending part of the movie. If the scene belongs to the final part, we ask the crowd if it is a spoiler (*Spoile Scenes* task); otherwise, we ask the crowd to order it with respect to the other scenes in the same class (*Order Scenes* task).

Tasks have been configured according to the following patterns:

– *Position Scene*: task and microtask types are both set as *Classify*, using a *StaticAgreement* pattern with threshold 3. Having 3 classes, a maximum number of 7 executions grants that one class will get at least 3 selections. Each microtask evaluates 1 scene.
– *Order Scene*: task type is *Order*, while microtask type is set as *Like*. Each microtask comprises two scenes of the same class. Using a *SortByLiking* pattern, we ask performers to select (Like) which scene comes first in the movie script. A rank aggregation pattern calculates the resulting total order upon task completion.
– *Spoiler Scene*: Task and microtask type both set as *Like*. A *StaticAgreement* pattern with threshold 3 (2 classes, maximum 5 executions) defines the consensus requirements. Each microtask evaluates 1 scene.

We experiment with two workflow configurations. The first (**P1**) defines a *batch* data flow between the *Position Scene* and *Order Scene* tasks, while the second configuration (**P2**) defines the same flow as *stream*. In both variants, the data flow between *Position Scene* and *Spoiler Scenes* is defined as *stream*.

The **P2** configuration features a dynamical task planning strategy for the the *Order Scenes* task, where the construction of the scene pairs to be compared in is performed every time a new object is made available by the *Position Scenes* task. A conservation policy in the *Order Scenes* data manipulator ensures that all the new scenes are combined with the one previously received.

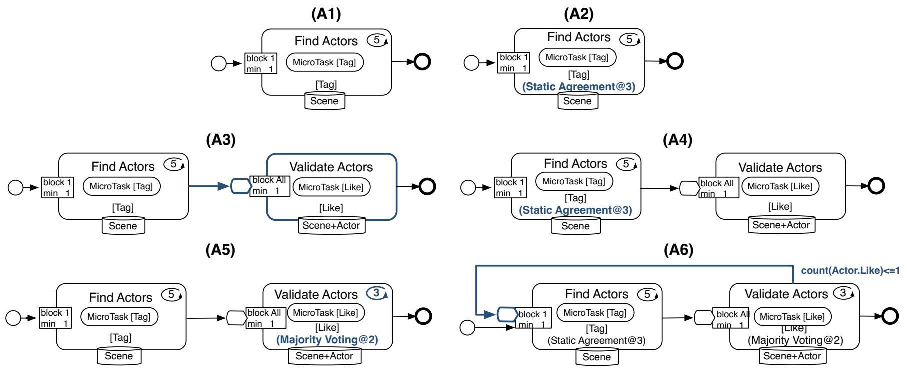


Fig. 8. Flow variants for the Actor scenario

6.2 Scenario 2: Actors

In the second scenario, we model a **create/decide** workflow pattern by asking the crowd to identify the actors that take part in the movie scenes; in *Find Actors*, performers indicate actors, in *Validate Actor* they confirm them. Tasks are designed as follows:

- *Find Actors*: Task and microtask types are set as *Tag*. Each microtask evaluates one scene; each scene is evaluated five times. Depending on the configuration, either no consensus pattern (**A1**, **A3**, **A5**) or a *StaticAgreement* pattern with threshold three (**A2**, **A4**, **A6**) is employed.
- *Validate Actors*: the task is preceded by a data manipulator function that transform the input Scene object and associated tags into a set of tuples $(Scene, Actor)$, which compose an object list subject to evaluation. In all configurations, microtasks are triggered if at least one object is available in the buffer. Note that each generated microtask features a different number of objects, according to the number of actors tagged in the corresponding scene. Configurations **A5** and **A6** features an additional *MajorityVoting* pattern to establish the final actor validation.

We tested this scenario with five workflow configurations, shown in Figure 8, and designed as follows:

- Configuration **A1** performs 5 executions and for each scene collects all the actors tagged at least once;
- Configuration **A2** performs 5 executions and for each scene collects all the actors tagged at least three times (StaticAgreement@3);
- Configuration **A3** adds the validation task to **A1**; the validation asks one performer to accept or reject the list of actors selected in the previous step;
- Configuration **A4** adds a validation task to **A3**, performed as in **A3**;
- Configuration **A5** is similar to **A3**, but the validation task is performed 3 times and a MajorityVoting@2 is applied for deciding whether to accept or not the object;

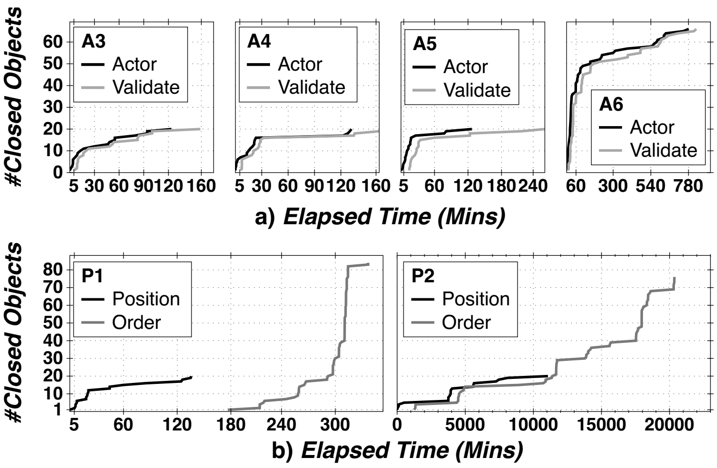


Fig. 9. Temporal distributions of closed objects

- Configuration **A6** extends **A5** by adding a StaticAgreement@3 on FindActors a feedback stream flow, originating from the *Validate Actors* task and directed to the *Find Actors* task, which notifies the latter about actors that were wrongly tagged in a scene (i.e., for which agreement on acceptance was not reached). Misjudged scenes are then re-planned for evaluation; for each scene, the whole process is configured to repeat until validation succeeds, or at most 4 re-evaluations are performed.

6.3 Results

We tested the performance of the described scenarios in a set of experiments performed on Amazon Mechanical Turk during the last week of September 2013. Table 1 and Table 2 summarise the experiment statistics for the two scenarios, 1700 HITS for a total cost of 39$.

Streaming Vs. Batch (Scenario 1: Positioning). In the first scenario we tested the impact on the application performance of the adoption of a stream data flow in a crowd workflow.

Time. Figure 9(b) shows the temporal distribution of closed objects for the **P1** and **P2** configurations. As expected, a stream flow (**P2**) allows for almost synchronous activation of the subsequent task in the flow, while batch scenario (**P1**) shows a strict sequential triggering of the second task. However, the overall duration of the workflow is not significantly affected by the change. While the first task of the flow behaves similarly in the two configurations, the second task runs significantly quicker in the batch flow, thus recovering the delay due to the sequential execution.

Quality. Table 3a shows the precision of the classification results of task *Position Scenes* (note that for this first part the two configurations are exactly the same, it makes no sense to compare the two results). Table 3b shows a measure of the quality of the obtained orders of scenes, i.e., Spearman's rank correlation coefficient of the resulting ranks from the *Order Scenes* task against the real order of scenes. Both tables show that the attained quality was not significantly influenced by the different task activation modes.

In summary, we didn't notice a different behaviour due to streaming. One possible reason is that in the batch configuration the entire set of assignments is posted at once on AMT, thus becoming more prominent in terms of number of available executions (and thus being preferred by performers, as widely studied [1]), while in a stream execution a small number of assignments is posted on AMT at every closing event of objects from the previous tasks.

Intra-Task Consensus Vs. Workflow Decision (Scenario 2: Actors). The second scenario aimed at verifying the impact that different intra-task and workflow patterns produced on the quality, execution time, and cost. We focused in particular on different validation techniques.

Time. Figure 9(a) and (c) shows the temporal distribution of closed object for configurations **A3-A6**. Configurations **A1** and **A2** are not reported because they are composed of one single task and thus their temporal distribution is not comparable. The temporal behaviour of the first and second tasks in the flow are rather similar (in the sense that the second one immediately follows the other). Validation is more delayed in **A5** due to the MajorityVoting pattern that postpones object close events. Configuration **A6** (Figure 9(c)) is significantly slower due to the feedback loop, which also generates a much higher cost of the campaign, as reported in Table 1. Indeed, due to the feedback, many tasks are executed several times before converging to validated results.

Quality. Table 4 reports the precision, recall and F-Score figures of the six configurations. The adoption of increasingly refined validation-based solutions (configurations **A3-A4-A5**) provides better results with respect to the baseline configuration **A1**, and also to the intra-task agreement based solution **A2**; validations do not have a negative impact in terms of execution times and costs. On the other hand, the complexity of of case **A6**, with the introduction of feedback, proved counter-productive, because the validation logic harmed the performance, both in monetary (much higher cost) and qualitative (lower results quality) senses, bringing as well overhead in terms of execution time. Notice

Table 3. Scenario 1 (Positioning), configuration P1 and P2: a) Precision of the *Position Scenes* classification task; b) Spearman's rank correlation coefficient of the resulting ranks from the *Order Scenes* task against the real order of scenes

(a)

Config.	**P Beg.**	**P Mid.**	**P End**
P1	0.50	1	0.11
P2	0.50	0.80	0.33

(b)

	Spearman Beg.	**Spearman Mid.**
P1	0.500	0.543
P2	0.900	0.517

Table 4. Scenario 2 (Actor): Precision, Recall, and F-score of the 6 configurations

	A1	**A2**	**A3**	**A4**	**A5**	**A6**
Precision	0.79	1	0.92	0.99	0.95	0.89
Recall	0.98	0.87	0.97	0.90	1	0.96
F-Score	0.85	0.91	0.93	0.93	0.97	0.90

that the configuration **A3** reaches the highest precision score. That's because the StaticAgreement strategy ensures that all the selected actors really appear in the image, while using the crowd for the validation part can add some errors (for instance some actors recognized in the *Find Actor* can be discarded in the *Validate Actors*). However note that the other configurations (**A3 - A5**) reach an higher recall and F-score value, meaning an overall better quality of the final result.

In summary, the above tests show an advantage of concentrating design efforts in defining better workflows, instead of just optimising intra-task validation mechanisms (based e.g. on majority or agreement), although overly complex configurations should be avoided.

7 Conclusions

We present a comprehensive approach to the modeling, design, and pattern-based specification of crowd-based workflows. We discuss how crowd-based tasks communicate by means of stream-based or batch data flows, and we define the option between loose and tight coupling. We also discuss known patterns that are used to create crowd-based computations either within a task or between tasks and we show how the workflow model is translated into executable specifications which are based upon control data, reactive rules, and event-based notifications.

A set of experiments demonstrate the viability of the approach and show how the different choices in workfllow design may impact on the cost, time and quality of crowd-based activities.

References

[1] Law, E., von Ahn, L.: Human Computation. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers (2011)

[2] Bozzon, A., Brambilla, M., Ceri, S.: Answering search queries with crowdsearcher. In: 21st Int.l Conf. on World Wide Web, WWW 2012, pp. 1009–1018. ACM (2012)

[3] Bozzon, A., Brambilla, M., Ceri, S., Mauri, A.: Reactive crowdsourcing. In: 22nd World Wide Web Conf., WWW 2013, pp. 153–164 (2013)

[4] Bernstein, M.S., Little, G., Miller, R.C., Hartmann, B., Ackerman, M.S., Karger, D.R., Crowell, D., Panovich, K.: Soylent: a word processor with a crowd inside. In: Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology, UIST 2010, pp. 313–322. ACM, New York (2010)

[5] Minder, P., Bernstein, A.: How to translate a book within an hour: towards general purpose programmable human computers with crowdlang. In: WebScience 2012, Evanston, IL, USA, pp. 209–212. ACM (June 2012)

[6] Doan, A., Ramakrishnan, R., Halevy, A.Y.: Crowdsourcing systems on the world-wide web. Commun. ACM 54(4), 86–96 (2011)

[7] Little, G., Chilton, L.B., Goldman, M., Miller, R.C.: Turkit: tools for iterative tasks on mechanical turk. In: HCOMP 2009, pp. 29–30. ACM (2009)

[8] Kochhar, S., Mazzocchi, S., Paritosh, P.: The anatomy of a large-scale human computation engine. In: HCOMP 2010, pp. 10–17. ACM (2010)

[9] Ahmad, S., Battle, A., Malkani, Z., Kamvar, S.: The jabberwocky programming environment for structured social computing. In: UIST 2011, pp. 53–64. ACM (2011)

[10] Marcus, A., Wu, E., Madden, S., Miller, R.C.: Crowdsourced databases: Query processing with people. In: CIDR 2011, pp. 211–214 (January 2011), `www.cidrdb.org`

[11] (OMG), O.M.G.: Business process model and notation (bpmn) version 2.0. Technical report (January 2011)

[12] Wang, J., Kumar, A.: A framework for document-driven workflow systems. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 285–301. Springer, Heidelberg (2005)

[13] Nigam, A., Caswell, N.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)

[14] Marcus, A., Wu, E., Karger, D., Madden, S., Miller, R.: Human-powered sorts and joins. Proc. VLDB Endow. 5(1), 13–24 (2011)

[15] Kazai, G., Kamps, J., Milic-Frayling, N.: An analysis of human factors and label accuracy in crowdsourcing relevance judgments. Inf. Retr. 16(2), 138–178 (2013)

[16] Little, G., Chilton, L.B., Goldman, M., Miller, R.C.: Exploring iterative and parallel human computation processes. In: Proceedings of the ACM SIGKDD Workshop on Human Computation, HCOMP 2010, pp. 68–76. ACM, New York (2010)

[17] Lin, C.H., Mausam, Weld, D.S.: Crowdsourcing control: Moving beyond multiple choice. In: UAI, pp. 491–500 (2012)

[18] Venetis, P., Garcia-Molina, H., Huang, K., Polyzotis, N.: Max algorithms in crowdsourcing environments. In: WWW 2012, pp. 989–998. ACM, New York (2012)

[19] Nowak, S., Rüger, S.: How reliable are annotations via crowdsourcing: a study about inter-annotator agreement for multi-label image annotation. In: Proceedings of the International Conference on Multimedia Information Retrieval, MIR 2010, pp. 557–566. ACM, New York (2010)

[20] Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. Journal of the Royal Statistical Society 39(1), 1–38 (1977)

[21] Davidson, S.B., Khanna, S., Milo, T., Roy, S.: Using the crowd for top-k and group-by queries. In: Proceedings of the 16th International Conference on Database Theory, ICDT 2013, pp. 225–236. ACM, New York (2013)

[22] Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering 17(6), 734–749 (2005)

[23] Alonso, O., Rose, D.E., Stewart, B.: Crowdsourcing for relevance evaluation. SIGIR Forum 42(2), 9–15 (2008)

SmartComposition: A Component-Based Approach for Creating Multi-screen Mashups

Michael Krug, Fabian Wiedemann, and Martin Gaedke

Technische Universität Chemnitz, Germany
{firstname.lastname}@informatik.tu-chemnitz.de

Abstract. The spread and usage of mobile devices, such as smartphones or tablets, increases continuously. While most of the applications developed for these devices can only be used on the device itself, mobile devices also offer a way to create a new kind of applications: multi-screen applications. These applications run distributedly on multiple screens, like a PC, tablet, smartphone or TV. The composition of all these screens creates a new user experience for single as well as for several users. While creating mashups is a common way for designing end user interfaces, they fail in supporting multiple screens. This paper presents a component-based approach for developing multi-screen mashups, named SmartComposition. The SmartComposition approach extends the OMELETTE reference architecture to deal with multiple screens. Furthermore, we enhance the OMDL for describing multi-screen mashups platform independently. We draw up several scenarios that illustrate the opportunities of multi-screen mashups. From these scenarios we derive requirements SmartComposition needs to comply with. A huge challenge we face is the synchronization between the screens. SmartComposition solves this through real-time communication via WebSockets or Peer-to-Peer communication. We present a first prototype and evaluate our approach by developing two different multi-screen mashups. Finally, next research steps are discussed and challenges for further research are defined.

Keywords: Mobile, distributed user interface, distributed displays, multi-screen applications, web applications, mashup, widgets.

1 Introduction

Internet-enabled devices are not anymore limited to computers and laptops. Devices, like smartphones, tablets or TVs, also offer users access to the Internet [8]. The number of mobile devices sold in 2013 exceeds the numbers of computers and laptops by factor three [10]. These new devices as well as the new capabilities of the Internet enable applications that can be used in several areas, such as entertainment, communication or productivity. While most of these applications can only be used on the device itself, mobile devices offer a way to create a new kind of applications: multi-screen applications. Multi-screen applications run distributedly on different devices, like a PC, smartphone, tablet or TV. The composition of these multiple screens creates a new user experience. Multi-screen applications are suitable for single as well as several users.

A particular kind of multi-screen applications are multi-screen mashups. These multi-screen mashups are also a special type of user interface mashups (UI mashups), the importance of which has increased significantly within the last years [1]. A mashup consists of several widgets that offer a limited functionality. By combining and aggregating these widgets more complex tasks can be solved. While each widget can work by itself, it has to be extended to support the communication with other widgets inside the mashup. Examples for these UI mashups are platforms like iGoogle or Yahoo! Pipes. The compositional way UI mashups are built eases the development of new rich web applications. Current approaches regarding UI mashups focus on applications that run on a single screen. They offer the opportunity to easily create end user interfaces [3] on different devices but not across these devices. Although the EU FP7 project OMELETTE has focused on telco service mashups, it proposed a reference architecture for UI mashups in [6,18]. This architecture facilitates the deployment of UI mashups to desktop as well as mobile devices, but cannot distribute the UI across several devices. Another outcome of this EU project was the Open Markup Description Language (OMDL) [18] which supports the process of designing, evolving and deploying UI mashups, but is also limited to single-screen mashups. Thus, we extend the OMELETTE reference architecture and the OMDL to deal with multi-screen mashups.

Developing web applications is a methodical process our approach needs to deal with. Therefore, aspects of software and web engineering have to be considered. The WebComposition approach [4] describes a way to design and develop web applications based on reusable components. These components can be exchanged or reused within other web applications. The purpose of this paper is to propose an approach to support end users in creating multi-screen mashups. Therefore, we want to extend the UI mashup approach to support multiple screens. We present an architecture that supports and eases the development process of multi-screen mashups.

The rest of the paper is organized as follows: In Section 2 we describe three scenarios to illustrate the use cases we focus on. From these scenarios we derive challenges that a solution has to deal with in Section 3. The SmartComposition approach is proposed in Section 4. We describe our extension of the OMELETTE reference architecture and the OMDL. A first prototype which follows the SmartComposition approach is demonstrated in Section 5. In Section 6 we evaluate our approach. We discuss related work in Section 7. Finally, we provide a conclusion and give an outlook towards future work in Section 8.

2 Scenarios

In this section we present three scenarios that illustrate where multi-screen applications can create a benefit for users. Using these scenarios we want to show what users could expect from such applications and what challenges exist that need to be solved.

Scenario 1. The first scenario focuses on media enrichment using multiple screens. While consuming a documentary on television, the user, called Amy, wants to see additional content about the currently watched show. This additional content could be a related article on Wikipedia, images from online photo services, like Google or Flickr, or where to buy items that were shown in the video. For easing the reading or enabling interaction with the additional content, it will be displayed on Amys smartphone or tablet, while she is still watching the documentary on her television. Figure 1 illustrates this scenario. While watching a documentary about the United Nations (cf. 1) the location of the UN headquarters in New York is shown on a map on Amys smartphone (cf. 2) as soon as the video reaches the scene where it is presented. In the same way other kinds of information, which are directly related to the current scene, are provided. Amy can easily decide what types of additional content she wants to receive. For deleting this information two use cases are applicable. First, the information about the UN headquarters' location will be removed from Amy's smartphone, if the documentary does not talk anymore about it. Second, by clicking on the information Amy can prevent the automatic removing of the information.

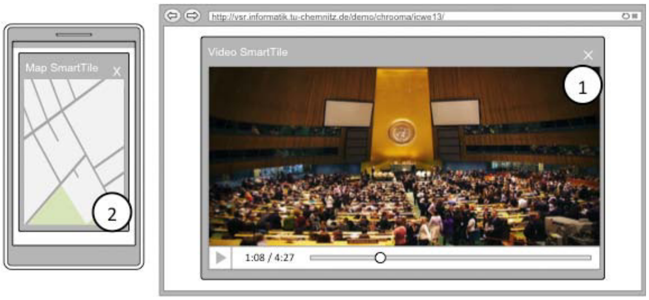


Fig. 1. Mockup illustrating media enrichment in Scenario 1

Scenario 2. The second scenario addresses the enhancement of a presentation by distributing additional information to multiple secondary screens. As an example, a professor is giving a talk in front of many students. While showing slides of his presentation via a projector, the students receive further information on their mobile devices. Thus, the students can read and see details about the current topic without messing up the slides with a lot of text. Furthermore, the students can use the additional information on their laptops as a script. The professor can keep his slides clean, while the students get a synced script related to the current lecture. While the professor is authenticated as a lecturer, he can broadcast slides and information. This functionality is denied to the students who do not have this privilege.

Scenario 3. Our third scenario aims at collaborative work on multiple screens. Therefore, we want to reuse a scenario described by Husmann et al. in [9]. In this scenario "two users, Alex and Bill, are planning a mountain biking trip at home in their living room". While Alex is planning the mountain biking route on her tablet, Bill is searching for railway connections on his smartphone. Both are using the television in their living room to share the results of their tasks on a larger display. When Alex chooses a start point for their mountain biking trip, this start point will be pushed to Bill's smartphone, where it is automatically inserted in the input field for the destination of his railway connection query. Bill can choose his favorite railway connections to be displayed on the television. Alex' chosen mountain biking route will be displayed as a map on the television next to the railway connection information.

3 Analysis

We use the presented scenarios in Section 2 to derive objectives our approach needs to deal with for creating multi-screen mashups. First, we begin with some basic findings. In all scenarios there is always a common context shared between all participating screens. This context has to be maintained. We identified two possible screen constellations. In the first one a primary screen publishes information to all secondary screens (cf. Scenarios 1 and 2). The secondary screens are only consuming and displaying information. An interaction with the information is possible, but without reflection to the original publisher. In the second constellation all screens are in an equal role (cf. Scenario 3). All of the participants are publishing and consuming information. A more interactive and collaborative usage of the application is possible. This setup requires a more complicated synchronization mechanism. Furthermore, there are also two scenario types regarding the number of users: single-user (cf. Scenario 1) and multi-user (cf. Scenarios 2 and 3). Both will require a different application design.

Based on the scenarios in Section 2, we now state several objectives that our approach should fulfill.

Simplicity of Mashup Creation. We targeted several scenarios in Section 2 where multi-screen mashups can provide a suitable solution. Thus, our approach should enable end users to easily create these multi-screen mashups for their specific tasks. This includes the reuse of developed widgets as well as the easy configuration of a set of devices which interact as a multi-screen. Inspired by the WebComposition approach from Gaedke et al. in [4,5] we apprehend widgets as loosely-coupled components, which were assembled in a mashup to a new application. Furthermore, the end users should not fiddle with designing UI mashups for mobile use only. The process of creating a multi-screen mashup is independent of the targeted device which can be a mobile as well as a PC or laptop.

Support for Multi-screen Usage. The focus of our proposal is the development of multi-screen mashups. Therefore, our approach should provide functionality to develop applications that can be used with more than one screen or device. Thus, there is a need for unique identification and data synchronization. The types of the participating screens should be detected and used to deliver an according user interface. While there can be a lot of devices which use the system, they should not be connected to any other available screen. This objective implies that devices can be grouped to so called workspaces and cannot communicate with devices in different workspaces. As we described in Scenarios 1 to 3 the support for multiple screen is the central point of our approach.

Support for Real-Time Communication and Synchronization. To propagate information without noticeable delay between server and client a component for real-time communication and synchronization should be available. We define the communication as real-time, when the latency of the transmission of a message from one widget to another one is 50 milliseconds or less [15]. That ensures no noticeable delay when distributing information across one screen as well as multiple screens. Furthermore, the components on one screen must be able to communicate among themselves.

Collaboration Support. The applications to develop should not only be used by a single user. We want to support the development of multi-screen multi-user mashups. Thus, our approach should provide the basis for collaborative work, such as user identification and data synchronization. This is highly related to the real-time synchronization requirement. Scenario 3 illustrates this objective best, while Scenario 2 also partially requires some support in collaboration.

Authentication. Nowadays it is important to provide a personalized user experience as well as social interaction. Thus it is required to offer the users a way to authenticate themselves. Using protocols like WebID even an authentication without requiring a password is possible. A user and role based identity management would provide a flexible way to handle most scenarios. The usage of multiple different devices creates a challenge to provide user-friendly login mechanisms for different use cases.

After we specified our objectives, we propose our own approach within the next section.

4 The SmartComposition Approach

To fulfill the requirements gathered from the analysis we propose a component-based approach to develop multi-screen mashups, called SmartComposition. We extend the OMELETTE reference architecture [18] for fitting the requirements in Section 3. In the following paragraphs we describe the SmartComposition architecture, which is depicted in Figure 2. We highlight our significant changes related to the OMELETTE reference architecture with underlined text. Our proposal consists of four parts: SmartComposition Runtime Environment, Multi-Screen Workspace, Information Store, SmartScreen. These parts interact like described in the following.

The SmartComposition Runtime Environment runs a multiple instances of a mashup. Thus, it handles multiple Multi-Screen Workspaces which are separated and secured against other existing Multi-Screen Workspaces. To enable the security between the Multi-Screen Workspaces we introduce the Session-Handler.

The Session-Handler knows all running workspaces and instantiates the Multi-Screen Workspaces. Each Multi-Screen Workspace contains a Workspace Manager.

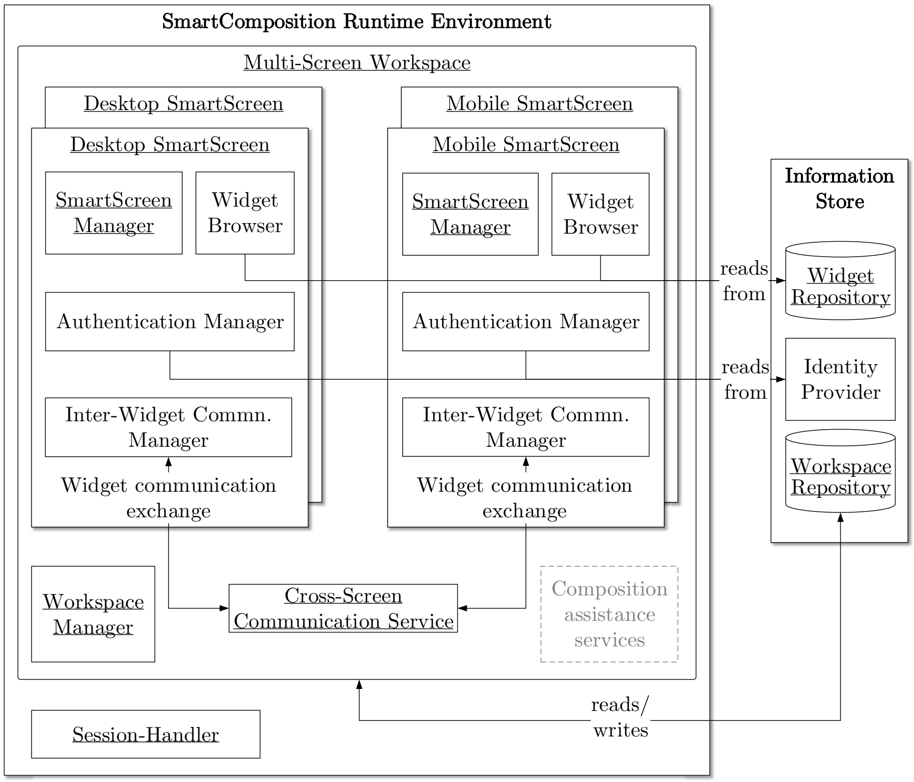


Fig. 2. Architecture of the SmartComposition approach

The Workspace Manager handles the workspace of a single user as well as several users who use the mashup in a collaborative way. Furthermore, the Workspace Manager is responsible for initialization of the SmartScreens and for the state management of the whole workspace. The illustrated component composition assistance services [16] in Figure 2 are not covered by our approach yet. Therefore, a workspace is defined as a set of one or more SmartScreens, which share the same information space and can communicate among themselves. Thus, the simplest form of a workspace consists of a single SmartScreen. A workspace is identified by a workspace ID and described by its workspace configuration. The configuration contains information about the SmartScreens which are connected, the widgets on each screen and additional properties. We use the OMDL as the basis for our configuration notation. Therefore, we extend the OMDL to support multi-screen aspects. How we did this and what changes we made is described later in this section.

The inter-widget communication is a highly important topic when developing mashups [1]. Our approach does not improve the inter-widget communication by itself but extends it for multi-screen mashups. On a single SmartScreen we use the publish-subscribe pattern for transmitting messages from one widget to one or several other widgets. For supporting inter-widget communication across multiple screens we extend the inter-widget communication component to publish the message to other connected SmartScreens (cf. Figure 3). That is, if a widget publishes a message on a certain topic, the inter-widget communication component publishes this message to all widgets on the same SmartScreen that are subscribed on the same topic. Furthermore, the message is transmitted to a Cross-Screen Communication Service where the message is sent to all connected SmartScreens within the same session. The assignment of SmartScreens and the workspace session they belong to is done by the Session-Handler. This ensures that a SmartScreen can only communicate with other SmartScreens of the same workspace. On each SmartScreen the inter-widget communication component publishes the message also to the subscribed widgets on their SmartScreen. The Cross-Screen Communication Service can work in two modes: broadcast mode and multi-layered publish-subscribe mode. In broadcast mode the Cross-Screen Communication Service sends every incoming message of one SmartScreen to all other SmartScreens of the same workspace. However, in the multi-layered publish-subscribe mode the inter-widget communication component of each SmartScreen subscribes to topics at the Cross-Screen Communication Service. The topics to subscribe depend on the topics that the widgets subscribe to on their SmartScreen. After publishing a message from a widget to the inter-widget communication component of its SmartScreen the message will be published to all subscribed widgets. Furthermore, the message will be published to the Cross-Screen Communication Service. There, the message will be published to inter-widget communication component of the SmartScreens, if they are subscribed to the topic. Each inter-widget communication component then publishes the message to the subscribed widgets.

As defined before, a workspace consists of SmartScreens. A SmartScreen is an abstract representation of a web browser window. It provides the runtime environment for all client-side components. Each SmartScreen has its own identifier. This identifier has to be at least unique in the workspace. This is important for the communication between the screens. The SmartScreen Manger within the SmartScreen initializes the widgets, handles the state management and provides access to device-specific properties. It detects the type and resolution of the device and uses this information to adapt the presentation. Thus, it is responsible for composition of the user interface.

The information store is the part which is dealing with all required and available persistent data. It consists of three components: the widget repository, the identity provider and the workspace repository.

Described in the OMELETTE reference architecture in [18] the widget repository is part of the SmartComposition Runtime Environment, but there are also meta data about the widgets available in the information store. Thus, we decided

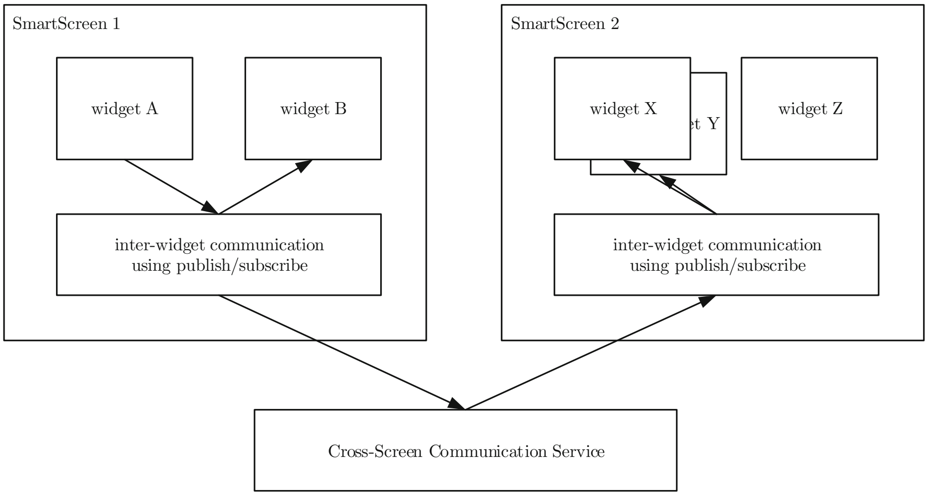

SmartScreen 1
widget A
widget B
inter-widget communication using publish/subscribe
SmartScreen 2
widget X
widget Y
widget Z
inter-widget communication using publish/subscribe
Cross-Screen Communication Service

Fig. 3. Cross-screen message flow of inter-widget communication

to combine both parts and move the widget repository to the information store. There it is responsible for delivering the executables as well as providing meta data about the widgets. The end user can search for a specific widget based on the name or id of the widget. Furthermore, the end user can discover widgets based on their meta data. That is, a widget can be selected by the messages it can consume or the way it presents the proposed information, such as a map widget which shows given coordinates on a map excerpt. The widget repository is also responsible for delivering the executables to the SmartComposition Runtime Environment. In case of W3C widgets [19] the executables contains HTML-, CSS- and JavaScript-files which can be delivered to the SmartScreen without much effort.

The identity provider offers information about the current user or users. It contains endpoints for verifying the user's identity. While our approach enables using different identity management concepts, we propose the usage of WebID [17]. When using WebID the identity provider gives access to the users' WebID profiles. The supplied information about the authenticated user can also be used for accessing resources from the widget repository as well as the workspace repository [7]. Closely related to the identity provider is the Authentication Manager within the SmartScreens.

Therefore, the Authentication Manager requests the users WebID certificate. The WebID URI is extracted from the Subject Alternative Name of the WebID certificate. Then, the Authentication Manager requests the WebID profile from the Identity Provider and verifies that the public key of the users WebID certificate is equal to the public key in the corresponding WebID profile. This is the authentication part within the WebID flow described in [17]. Afterwards, based on the users WebID, authorization could be granted. Based on the authenticated user access to restricted resources can be permitted or denied, such as the users workspace or some special widgets, which are limited to specific users. To ease the access to a users roles, the Authentication Manager offers an interface to request the users WebID profile for personal information, like name, birthdate and more. Furthermore, the users roles can be requested by other widgets or core components of the SmartComposition Runtime Environment. However, the users relationships can be queried without requesting the users WebID profile again.

The workspace repository stores and offers predefined or user-specific Multi-Screen Workspaces. Predefined workspaces will be used if there is no user-specific workspace existing. Thus, our approach enables the developer to define the default appearance of a single SmartScreen. Defining a default workspace with more than one SmartScreen is not useful, because the developer cannot predict with how many devices the user will use the application. However, it is possible to define multiple different default workspaces regarding to context specific information, such as the users role, device or geo-location. Within the default workspace a set of widgets and their position on the SmartScreen is defined.

When a user or several users are using the multi-screen web application, it could be necessary to save the current Multi-Screen Workspace. The users workspace will be saved by storing the OMDL description of the workspace in the workspace repository. While a Multi-Screen Workspace can contain multiple SmartScreens, it is at least necessary that one device wants to restore its SmartScreen as part of the workspace. Therefore, the device needs to save at least the workspace identifier within its client-side storage. The workspace repository offers an interface where the OMDL description corresponding to a given identifier is responded. From this OMDL description the device can extract its workspace configuration and can so restore the widgets on this SmartScreen. When another device of this workspace wants to restore its SmartScreen, it also sends the workspace identifier and retrieves the OMDL description of the workspace where it can extract its SmartScreen. On restoring the second or another SmartScreen the communication channel between these SmartScreen will be established by the Cross-Screen Communication Service.

4.1 Extension of OMDL

We use the OMDL for describing our Multi-Screen Workspaces and for storing them in the workspace repository. While the OMDL defines three levels for describing mashups [18], we just use the Physical Level for the SmartComposition approach. However, the defined Physical Level needs to be extended to support multiple screens, because the OMDL has no support for distributed mashups. The OMDL uses XML documents to describe mashups, their layout and the apps they contain. The root element of an OMDL document is the workspace. In the SmartComposition approach we apply the OMDL workspace to theMulti-Screen Workspace described above.

While the OMDL does not support multiple screens yet, we extend the vocabulary of the OMDL by adding several elements. The first new element is the SmartScreen which is added within the workspace element. The SmartScreen element is used to define a SmartScreen within a Multi-Screen Workspace. Thus, there can be multiple SmartScreen elements. The SmartScreen element has an attribute id, which is unique and identifies the SmartScreen. Furthermore, the SmartScreen element has a child element named type, which defines the type of the device the SmartScreen runs on. For dealing with collaborative multi-user workspaces we define a user element within the SmartScreen element, which refers to the authenticated users WebID URI.

```xml
<workspace xmlns="http://omdl.org/">
 <identifier>http://example.org/workspacerepo/ws1</identifier>
 <title>Media</title>
 <date>2012-07-03T14:23+37:00</date>
 <SmartScreen id="phone1">
   <type>Smartphone</type>
   <user>http://example.org/people/alice.rdf#me</user>
   <grid><height>4</height><width>2</width></grid>
 </SmartScreen>
 <SmartScreen id="pc1">
   <type>PC</type>
   <user>http://example.org/people/bob.rdf#me</user>
   <grid><height>6</height><width>10</width></grid>
 </SmartScreen>
 <app id="http://example.org/workspacerepo/ws-of-alice/1">
   <SmartScreen>phone1</SmartScreen>
   <type>MAP</type>
   <link rel="source" href="http://example.org/repo/w1"
     type="application/widget"/>
   <position><x>0</x><y>0</y></position>
   <size><height>2</height><width>2</width></size>
 </app>
 <app id="http://example.org/workspacerepo/ws-of-alice/2">
   <SmartScreen>pc1</SmartScreen>
   <type>LIST</type>
   <link rel="source" href="http://example.org/repo/w2"
     type="application/widget"/>
   <position><x>0</x><y>0</y></position>
   <size><height>5</height><width>2</width></size>
 </app>
</workspace>
```

Listing 1.1. Example of a workspace description in OMDL

Each SmartScreen has an individual fine grained grid to arrange the widgets. To store the position of the widgets the SmartScreen element has a child element named grid. This element has two child elements height and width which represent the number of rows and columns the grid has. Another element we adapted is the app element. In OMDL this element describes a part of the mashup. In the

SmartComposition approach we define an app as a widget. While the OMDL has no multi-screen support yet, we also need to extend the app element to define on which SmartScreen the widget is located. Therefore, we introduce a SmartScreen element within the app element. This SmartScreen element contains the id of the containing SmartScreen. Furthermore, we assume that the type element of the app can be extended by several types which are required for a specific multi-screen web application. However, the alignment of apps in OMDL does not fit our approach, which focuses on a finer grained positioning. Therefore, we extended the position element by two new elements which are x and y. These elements represent the position of the widget regarding to a fine grained grid on the SmartScreen, where x means the distance to the left side and y means the distance to the top. Furthermore, we added a new element to the app element named size. This element has two child elements height and width, which mean the size of the widget regarding to fine grained grid on the SmartScreen. An example of a SmartComposition workspace described in OMDL can be seen in Listing 1.1.

5 Prototype

To validate the SmartComposition approach we have implemented a first prototype, which is based on another work we recently presented at the ICWE 2013 [12,14] and WWW 2014 [13]. The prototype demonstrates the Scenarios 1 and 2 we described in this paper (cf. Figure 4). All the client-side components we describe are implemented in JavaScript running as a web application in a web browser.

To achieve a multi-screen experience we implemented the following components. We created a SmartScreen class. This class works as described in our approach and is the host for the widgets. For our user interface components we implemented a basic widget class which provides the interface and basic functionality. This basic class is used to derive various new widget types using prototype-based inheritance. We have implemented widgets that can display a map, images from different web sources, excerpts from Wikipedia, text, translations or tweets. The widgets can be added and removed on runtime. The user can arrange them in a grid-based layout by using drag-and-drop.

To handle the information exchange between the widgets we implemented a component as part of the inter-widget communication. This component provides the publish-subscribe pattern and thus offers loosely coupled communication. Once a widget is added to the SmartScreen it can subscribe to one or more topics on which events are published.

The workspace configuration of the SmartScreens is currently stored on the client-side using the HTML5 feature LocalStorage. Using the proposed workspace identifier the users customized arrangement can be restored. A workspace repository is not yet implemented.

To make the prototype work on multiple screens we extended the inter-widget communication component with a synchronization mechanism. This mechanism uses WebSockets to propagate the events which were published on one

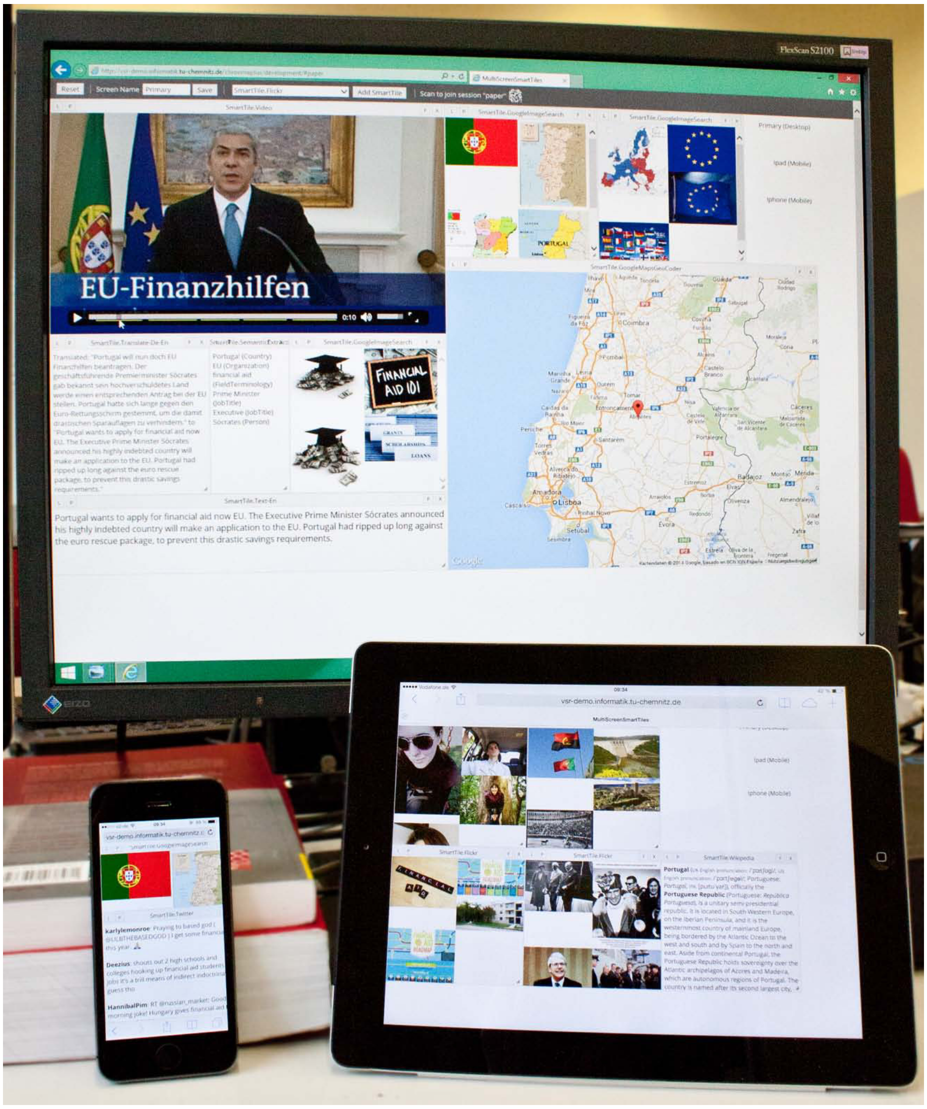

EU-Finanzhilfen
Portugal wants to apply for financial aid now EU. The Executive Prime Minister Sócrates announced his highly indebted country will make an application to the EU. Portugal had ripped up long against the euro rescue package, to prevent this drastic savings requirements.
vsr-demo.informatik.tu-chemnitz.de
FlexScan S2100

Fig. 4. Prototype of a multi-screen mashup described in Scenario 1. On the primary screen (top) a german news cast is played. Meta data from this video are published to all widgets within the workspace. Additional information are displayed on the different screens, such as Google Maps on the primary screen, Flickr images on the iPad (bottom right), and Twitter feed on the iPhone (bottom left).

SmartScreen to the other connected screens. This extension also handles the re-publishing of received events to the widgets. Thus, the whole communication is synchronized with all connected SmartScreens. This approach assures that each client behaves equally regardless of where the event was published. To make that work we also had to implement a server-side component. Once a new SmartScreen is added to a workspace the RTC components registers the screen at the Session-Handler using the workspace ID, the SmartScreens own identifier and device specific information. This data is supplied to the other participants.

To deal with the Cross-Screen Communication Service we implemented a WebSockets server that provides methods for propagating information to other connected clients. It handles the propagation of the events it receives to all SmartScreens that are combined in a workspace. Using the workspace identifier the related SmartScreens are selected. This component is implemented as a Node.JS application.

Demonstration. The prototype presented in this paper is available for testing at: `http://vsr.informatik.tu-chemnitz.de/demo/chrooma/icwe14/`

6 Evaluation

In this section we evaluate the SmartComposition approach. Based on the analysis in Section 3, we examine how the approach assists in developing multi-screen mashups. We outline our expectations and explain actual findings.

A big challenge our approach has to deal with is the support of multiple screens. Therefore, we introduced the concept of workspaces where several SmartScreens create the interface of the mashup. The introduced parts of our architecture, the Cross-Screen Communication Service and the Session-Handler, enable the inter-widget communication across multiple screen (cf. Figure 2). This two parts also ensure that only screens within the same workspace can communicate with each other. Thus, the workspaces are isolated from each other. We extended the OMDL to support multiple screens within a workspace and used it to store and restore the configuration of a workspace.

Real-time communication for the SmartScreens has to be enabled for synchronization and communication of multiple SmartScreens within a workspace. While web systems often use long-polling for nearly real-time communication, we considered that new technologies, such as WebRTC or WebSockets, which are introduced with HTML5, might better correspond to our requirements. Therefore, we designed our RTC component using WebRTC and WebSockets and support long-polling as a fallback mechanism for browsers, which do not support these new technologies. We evaluate the performance of the RTC component in our prototype by measuring the duration of transporting an information from one widget to another. A special widget was developed, which publishes a timestamp to a second widget. The second one re-publishes the timestamp to its source. After receiving the timestamp on the first widget the difference between the current timestamp and the received one is calculated and then halved. We examined different constellations. The first one was where both widgets were

Table 1. Transport duration of a message from one widget to another in milliseconds

	Average	**Standard Deviation**
Same SmartScreen	5,121	0,654
Same PC	4,100	0,921
PC - iPad	38,735	26,842
PC - PC (WiFi)	26,668	19,036
PC - PC (Wired)	3,494	1,254

on the same SmartScreen. Another one was where the widgets were on different SmartScreens which run on the same PC. The third constellation was one SmartScreen on a PC and the other on a WiFi-connected iPad. The fourth one was two SmartScreens on two PCs which are connected via WiFi. Two PCs are connected via wire in the last constellation. The statistical values of our evaluation can be seen in Table 1. As we proposed in Section 3 the real-time communication is satisfied as the transport of a message from one widget to another is less than 50 ms. Our findings show that the largest latency occurs at the communication between a widget on the PC and a widget on the iPad with 38,735 ms. We can argue the difference in the average communication between the same SmartScreen and two SmartScreens on the same PC with the multi-threading architecture of the PC. That is, with two SmartScreens in two browser windows enables more parallelism in execution, because each browser window can be run in a different thread. The high standard deviation in the communication via WiFi (PC-iPad and PC-PC) can be ascribed to some side effect caused by the WiFi. To ensure that the higher latency when using WiFi does not is caused by constant term we give the relation of the IP-latency measured by ping in Table 2.

Table 2. Ping latency between two PCs in milliseconds

	Average	**Standard Deviation**
PC - PC (WiFi)	3,800	1,279
PC - PC (Wired)	1,025	0,798

We evaluate the simplicity of creation by developing two mashups for Scenario 1 and Scenario 2. In Scenario 1 we developed some new widgets for displaying a video and processing the meta data, such as the given subtitles. Other widgets could be reused in both scenarios, like a Wikipedia-widget which is displaying an article from Wikipedia or a GoogleMaps-widget which shows a map excerpt. Since the separation of the SmartScreens into their workspaces is an essential part of our approach, the end user has not to deal with the configuration of the workspaces. A mashup created based on our approach runs on PCs, laptops, smartphones, tablets as well as on web-enabled TVs.

For supporting authentication we propose a two tier approach The Authentication Manager within the SmartComposition Runtime Environment deals with the process of authentication and offering interfaces for accessing roles and permissions. The Identity Provider in the Information Store gives access to the users' profile and personal data, such as name, birth date or e-mail address. The usage of WebID offers the opportunity to let users easily login with their different devices. They can use a WebID certificate on each device which refers to the same WebID profile of the user.

Performing collaborative tasks is facilitated by multi-user workspaces and real-time communication. For collaborative work on a multi-screen web application we extended the OMDL to define a specific user for each SmartScreen. Thus, a device, which is authenticated by the users WebID, receives the corresponding workspace and can load the SmartScreen associated to the device. This enables several users to use the same workspace. Utilizing the real-time communication functionality of the RTC component synchronization of different users SmartScreens can be achieved.

7 Related Work

Our approach is closely related to work in three research domains: distributing and migrating Web UIs, data mashups, and classic user interface mashups (UI mashups). The DireWolf approach proposed in [11] enables the distribution and migration of Web widgets between multiple devices. While our approach focuses on developing multi-screen mashups from scratch, the DireWolf approach extends the functionality of single-device mashups to run across multiple devices. The framework is involved in every layer of a widget-based Web Application: the widget itself, the client browser, the backend service and the data storage. DireWolf manages communication between widgets on one device as well as between widgets on multiple devices. It also extends the functionality of common widget spaces with a shared application state. While our approach uses HTML5 features, such as WebSockets, for communication the DireWolf approach uses the XMPP protocol and its publish-subscribe extensions. That is, the DireWolf approach uses an additional layer within the communication architecture, which increases the effort for maintaining and exchanging parts of this communication layer. Furthermore, the usage of XMPP can increase the incompatibility when using mobile devices inside restricted GSM networks.

MultiMasher is a visual tool to create multi-device mashups from existing web content [9]. It aims at designing mashups without the need for modeling or programming. The MultiMasher runs in the browser, connects to the MultiMasher server and provides the user with a toolbar. After the user has connected his devices he can load any web site, which he can then mashup. UI elements can be selected visually and distributed (move or copy) to other connected devices. The mashups can be saved, loaded and reused as basis for other mashups. MultiMasher uses event forwarding to propagate changes to the connected clients and only displays the previously selected UI elements on each device. MultiMasher focuses on the creation of mashups and not on the creation of new applications.

The EU FP7 Project OMELETTE is closely related to our work, because it proposes a reference architecture for designing platforms for UI mashups [18]. They focus on end users with less or no programming skills to create their own mashups for fulfilling a certain goal. The outcome of this project was a prototype based on Apache Rave and Apache Wooky [2]. After a user log in, she can restore recent workspaces or create a new one by adding widgets to her workspace. The focus of the OMELETTE project was on the automatic composition of these mashups. Therefore, they invented some great strategies for suggesting widgets to add or for establishing the inter-widget communication between two widgets based on the user's behavior. While the OMELETTE reference architecture is a well-proven approach, it lacks in supporting UI mashups which run distributed across multiple screens. Thus, we extend the proposed reference architecture to support workspaces which include multiple screens. We also showed that concepts of inter-widget communication designed for a workspace on one screen works also for a workspace across multiple screens.

Mashup tools, like Yahoo! Pipes, offer end users developing data mashups. End user in this context means people with no or less development skills. While our approach focuses on user interface mashups, Yahoo! pipes is focusing on mixing popular data feeds to create data mashups via a visual editor. Therefore, it lets users transform, aggregate, transform and filter one or more data sources, such as RSS/Atom feeds or XML sources, and output this as a RSS feed. Thus, it enables end users to develop a kind of business logic for processing data [20]. However, Yahoo! Pipes does not offer a multi-screen environment where the created mashups can be executed.

8 Lessons Learned and Outlook

The SmartComposition approach proposed in this paper enables end users to easily create multi-screen mashups. We extended the OMELETTE reference architecture to deal with mashups across multiple screens. The Cross-Screen Communication Service extends classic inter-widget communication to work on a trans-screen level, while separating different workspaces from each other via the Session-Handler. Furthermore, our approach handles the challenge of real-time communication. The component-based architecture enables adding required and removing obsolete components. Thus, SmartComposition can be adjusted for different multi-screen web application scenarios. Both, the workspace repository as well as the widget repository offer a generalized access to widgets and workspaces.

Our future work will focus on integrating commonly used widget formats, such as W3C-widgets or Opera widgets. We also plan to evolve the cross-screen inter-widget communication by considering constraints given by several roles. That is, in a multi-user scenario, e.g., Scenario 3, one user is permitted to update a specific widget on a shared screen, while another user just has read-only access to the presented information. Another aspect we want to focus on in future work is the authorization of widgets to use the users' external services, such as Facebook or Google Drive.

As described in Section 4 the composition assistance services, such as automatic composer and workspace pattern recommender, are not yet a part of our approach for multi-screen mashups. We plan to adapt the approaches proposed by Roy Chowdhury in [16] for multi-screen mashups. Different use cases in this field of research are possible: The user gets a recommendation to add an additional screen, like a smartphone or a laptop, when she is using the mashup to accomplish her desired goal in a better way. When a user starts a new workspace, the widgets will be deployed to all available screens. This can be done by learning from the behavior of the users or by designing a default workspace.

Acknowledgment. This work was supported by the Sächsische Aufbaubank within the European Social Fund in the Free State of Saxony, Germany (Project Crossmediale Mehrwertdienste für die digitale Mediendistribution).

References

1. Chudnovskyy, O., Fischer, C., Gaedke, M., Pietschmann, S.: Inter-widget communication by demonstration in user interface mashups. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 502–505. Springer, Heidelberg (2013)
2. Chudnovskyy, O., Nestler, T., Gaedke, M., Daniel, F., Fernández-Villamor, J.I., Chepegin, V., Fornas, J.A., Wilson, S., Kögler, C., Chang, H.: End-user-oriented telco mashups: the omelette approach. In: Proceedings of the 21st International Conference Companion on World Wide Web, pp. 235–238. ACM (2012)
3. Chudnovskyy, O., Pietschmann, S., Niederhausen, M., Chepegin, V., Griffiths, D., Gaedke, M.: Awareness and control for inter-widget communication: Challenges and solutions. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 114–122. Springer, Heidelberg (2013)
4. Gaedke, M., Rehse, J.: Supporting compositional reuse in component-based web engineering. In: Proceedings of the 2000 ACM Symposium on Applied Computing, vol. 2, pp. 927–933. ACM (2000)
5. Gaedke, M., Turowski, K.: Specification of components based on the web-composition component model. In: Managing Information Technology in a Global Economy, p. 411 (2001)
6. Gebhardt, H., Gaedke, M., Daniel, F., Soi, S., Casati, F., Iglesias, C., Wilson, S.: From mashups to telco mashups: A survey. IEEE Internet Computing 16(3) (2012)
7. Hollenbach, J., Presbrey, J., Berners-Lee, T.: Using rdf metadata to enable access control on the social semantic web. In: Proceedings of the Workshop on Collaborative Construction, Management and Linking of Structured Knowledge (CK 2009), vol. 514 (2009)
8. Horizont: Report TV-Marketing, Ausgabe 17, p. 40 (April 2012), http://www.horizont.net/report
9. Husmann, M., Nebeling, M., Norrie, M.C.: MultiMasher: A visual tool for multi-device mashups. In: Sheng, Q.Z., Kjeldskov, J. (eds.) ICWE Workshops 2013. LNCS, vol. 8295, pp. 27–38. Springer, Heidelberg (2013)
10. IDC Coroporate USA: Tablet Shipments Forecast to Top Total PC Shipments in the Fourth Quarter of 2013 and Annually by 2015, According to IDC (2013), http://www.idc.com/getdoc.jsp?containerId=prUS24314413

11. Kovachev, D., Renzel, D., Nicolaescu, P., Klamma, R.: DireWolf - distributing and migrating user interfaces for widget-based web applications. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 99–113. Springer, Heidelberg (2013)
12. Krug, M., Wiedemann, F., Gaedke, M.: Media enrichment on distributed displays by selective information presentation: A first prototype. In: Sheng, Q.Z., Kjeldskov, J. (eds.) ICWE Workshops 2013. LNCS, vol. 8295, pp. 51–53. Springer, Heidelberg (2013)
13. Krug, M., Wiedemann, F., Gaedke, M.: Enhancing Media Enrichment by Semantic Extraction. In: Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion, pp. 111–114. International World Wide Web Conferences Steering Committee (2014)
14. Oehme, P., Krug, M., Wiedemann, F., Gaedke, M.: The chrooma+ approach to enrich video content using html5. In: Proceedings of the 22nd International Conference on World Wide Web Companion, pp. 479–480. International World Wide Web Conferences Steering Committee (2013)
15. Pantel, L., Wolf, L.C.: On the impact of delay on real-time multiplayer games. In: Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video, pp. 23–29. ACM (2002)
16. Roy Chowdhury, S., Chudnovskyy, O., Niederhausen, M., Pietschmann, S., Sharples, P., Daniel, F., Gaedke, M.: Complementary assistance mechanisms for end user mashup composition. In: Proceedings of the 22nd International Conference on World Wide Web Companion, pp. 269–272. International World Wide Web Conferences Steering Committee (2013)
17. Sporny, M., Inkster, T., Story, H.: WebID 1.0: Web Identication and Discovery (2011), `http://www.w3.org/2005/Incubator/webid/spec/`
18. University of Trento: D2.2 - Initial Specification of Mashup Description Language and Telco Mashup Architecture. Tech. rep., University of Trento (2011)
19. W3C: Packaged Web Apps (Widgets) - Packaging and XML Configuration, 2nd edn. (2012), `http://www.w3.org/TR/2012/REC-widgets-20121127/`
20. Yu, J., Benatallah, B., Casati, F., Daniel, F.: Understanding mashup development. IEEE Internet Computing 12(5), 44–52 (2008)

SSUP – A URL-Based Method to Entity-Page Discovery

Edimar Manica[1,2], Renata Galante[2], and Carina F. Dorneles[3]

[1] Campus Avançado Ibirubá - IFRS - Ibirubá, RS, Brazil
edimar.manica@ibiruba.ifrs.edu.br
[2] PPGC - INF - UFRGS - Porto Alegre, RS, Brazil
galante@inf.ufrgs.br
[3] INE/CTC - UFSC - Florianópolis, SC, Brazil
dorneles@inf.ufsc.br

Abstract. Entity-pages are Web pages that publish data representing one only instance of a certain conceptual entity. In this paper we propose **SSUP**, a new method to entity-page discovery. Specifically, given a sample entity-page from a Web site (e.g., `Jolyon Palmer` entity-page from `GP2` Web site) we aim to find all same type entity-pages (driver entity-pages) from this Web site. We propose two structural URL similarity metrics and a set of algorithms to combine URL features with HTML features in order to improve the quality results and minimize the number of downloaded pages and processing time. We evaluate our method in real world Web sites and compare it with two baselines to demonstrate the effectiveness of our method.

Keywords: entity-pages, structural similarity, URL features, HTML features.

1 Introduction

The Web contains an increasing number of Web sites that can be considered a repository of pages with valuable information about real world entities. These pages are called *entity-pages* (or object-pages). Weninger et al. [8] define an entity-page as a Web page that describes a specific entity. For example, a Web page that describes a GP2 driver or a city council member.

Making use of the data presented in the entity-pages is an opportunity to create knowledge useful to several real applications (such as: comparative shopping, vertical search, named entity recognition and query suggestion). For instance, when we type the query "`players of real madrid`" on Google[1], it shows a list with all the players of Real Madrid Football Club including the attributes: `name`, `position` and `image`. If we click in one player of this list, the original query is replaced by the player name. The data for this kind of suggestion can be extracted from Wikipedia[2]. However, the Wikipedia does not contain all entities.

[1] `http://www.google.com/`
[2] `http://en.wikipedia.org/`

For example, the council members of Natal (a Brazilian state capital) are not presented in Wikipedia, but there is an official Web site with an entity-page for each council member describing the attributes: `name`, `political party`, `image`, `phone`, among others. As in general the entity-pages in the same Web site share a common template, HTML patterns can be learned to extract their data.

One important problem in this context is how to discover the entity-pages of the same type in one Web site. This is not a trivial task due to the variety of Web sites and structures. Some sites contain a page with a link to all entity-pages, while others divide these links in pages organized in a hierarchical manner and others divide these links in discrete pages (pagination). Moreover, some entity-pages have more details about an entity while others have less influencing in the HTML structure of the entity-pages. On the other hand, how the data in the entity-pages can change and new entity-pages can become available, it is necessary to re-execute the method to entity-page discovery periodically. Then, the number of downloaded pages and the processing time are important metrics to choose an effective method.

We found some methods to entity-page discovery in the literature. Methods based on HTML-tables [7] or human-compiled encyclopedias [6] are very restrictive. The method proposed by Weninger et al. [8] is based on the HTML and visual features, but in order to analyze the visual information is necessary to render the page in a browser loading all images, css and javascripts, then increasing the processing time. On the other hand, the method proposed by Blanco et al. [2] uses only the HTML features, then collect the information required is faster, but being based on only one kind of feature makes the method very sensitive to the parameter configuration.

Our goal is to find the set of entity-pages of the same type given a sample entity-page minimizing the number of downloaded pages and the processing time. For example, given the entity-page about `Jolyon Palmer` on the `GP2` Web site, we intent to find the set of driver entity-pages on the `GP2` Web site. The main contributions of this paper can be summarized as follows: (i) two structural URL similarity metrics ($weaksim_{url}$ and $strongsim_{url}$); (ii) **SSUP**: a new method to entity-page discovery that combines URL features with HTML features; (iii) show through experiments that combining URL features with HTML features improves the quality results of entity-page discovery and decreases the number of downloaded pages and the processing time; (iv) create datasets from real World Web sites for experiments.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 specifies the key concepts at the basis of our method. Section 4 presents the two structural URL similarity metrics proposed and the **SSUP** method. Section 5 presents a set of experiments we have conducted to evaluate our method on real Web sites comparing it with two baselines. Section 6 concludes the paper and presents future directions.

2 Related Work

Yu et al. [9] uses an SVM-based method to classify Web pages according to features from the content and URL of a Web page. Blanco et al. [3] presents a method for structurally clustering Web pages using only the URLs of the Web pages and simple content features. Although both methods analyze the URL of the Web page, their assumptions are different from **SSUP**, since they have as input all Web pages of the Web site in order to classify or cluster them while **SSUP** wants to download the minimum number of Web pages as possible to discover the entity-pages. **SSUP** represents URL in a way very similar to Blanco et al. [3], but **SSUP** treats this representation using the principle of TF-IDF [1] while Blanco et al. [3] uses the principle of minimum description length [4].

Kaptein et al. [6] present a Wikipedia-based method for entity-page discovery searching the Wiki-page for a link to the entity's home page. Lerman et al. [7] present a method that relies on the common structure of many Web sites, which present information as a list or a table, with a link in each entry pointing to a detail page containing additional information about that item. **SSUP** differs from these methods because it depends neither on the specific HTML markup nor on a human-compiled encyclopedia. He et al. [5] focus on deep Web, then the entity-pages are found through HTML form submissions while **SSUP** focuses on surface Web, then the entity-pages are found through browsing the Web site by its hyperlinks.

`Indesit` [2] and `GPP` (Growing Parallel Paths) [8] aim to find the set of entity-pages of the same type given a sample entity-page. `Indesit` models Web pages in terms of the DOM-structure of the HTML of the Web page and measure the structural similarity between two Web pages with respect to this feature. `GPP` combines both DOM-structure of the HTML and visual information. `SSUP` combines both DOM-structure of the HTML and URL information that is more robust that consider only DOM-structure of HTML and less costly that consider visual information. We choose `Indesit` and `GPP` as our baselines in the experiments because they are the most similar methods to **SSUP** and they depend neither on the specific HTML markup nor on a human-compiled encyclopedia. It is important to note that we reuse the definitions of `Indesit` related to the HTML of the Web pages because our contribution is how to use the URL features and to combine them with HTML features. The use of HTML features to entity-discovery seems to be consolidated since it was proposed in `Indesit` and reused by `GPP`.

3 Definitions

Here, we specify the key concepts at the basis of our method. First, we define the basic structures of a Web site (Section 3.1). Finally, we define the basic structures of a Web page (Section 3.2).

3.1 Web Site

A Web site is a directed graph whose nodes correspond to the pages of the Web that exist within the same domain and the edges correspond to the hyperlinks. According to our intuition, a part of a Web site is designed to allow the user to navigate in it from homepage until the entity-pages through a logical hierarchy of topics. The pages that are part of this logical hierarchy are classified as entity-pages or index-pages. Entity-pages were defined in Section 1. Index-pages are pages with links to the entity-pages or with links to other index-pages, whose role is to group entity-pages/index-pages in order to allow the user to navigate through a logical hierarchy of topics.

Definition 1. *(*Index page*) Let p be a Web page in the Web site s, EP be the set of all entity-pages in s, sEP be a subset of EP, IP be the set of all index-pages in s, sIP be a subset of IP. p is an index-page if it contains at least one link for each entity-page in sEP (or index-page in sIP) and its functional role in s is group sEP (or sIP).*

Each index-page and entity-page has a degree of topic specificity that represents the number of attributes that the page restricts. Entity-pages have the highest degree of topic specificity, since always entity-pages restrict more attributes that their index-pages, i.e., they represent the more specific topic in the logical hierarchy.

Definition 2. *(*Degree of topic specificity*) Let an entity e with two attributes a_1 and a_2. Let $P_x[w]$ ($P_y[z]$) be an index-page with links to entity-pages that describe instances where the value of a_1 is x (y) and the value of a_2 is w (z). Let P_x be an index-page with links to index-pages with links to entity-pages that describe instances where the value of a_1 is x, then we say that $P_x[w]$ and $P_y[z]$ has the same degree of topic specificity (because they restrict the same number of attributes) and $P_x[w]$ has higher degree of topic specificity that P_x (because $P_x[w]$ restricts more attributes that P_x).*

Example 1. In Figure 1, `Page A` has the lowest degree of topic specificity restricting the value of the attribute `type` to `car`. The pages `B`, `C` and `D`, besides restricting the `type` attribute value, restricts the value of the `manufacture year` attribute, then they have a degree of topic specificity higher than `Page A`. The pages `E`, `F`, `G`, `H`, `I` and `J` have the highest degree of topic specificity because they are entity-pages.

The logical hierarchy of topics is a tree, named entity-tree, whose root is the index-page with the lowest degree of topic specificity, the leaves are the entity-pages and non-leaves are index-pages.

Definition 3. *(*Entity-tree*) Let Et be a tree where the node set and the edge set are subsets of that of a given Web Site. Et is an entity-tree if it satisfies the following properties: (1) all leaf nodes are* entity-pages *of the* same type*; (2) all leaf nodes are at the same level; (3) all non-leaf nodes are index-pages; (4) the*

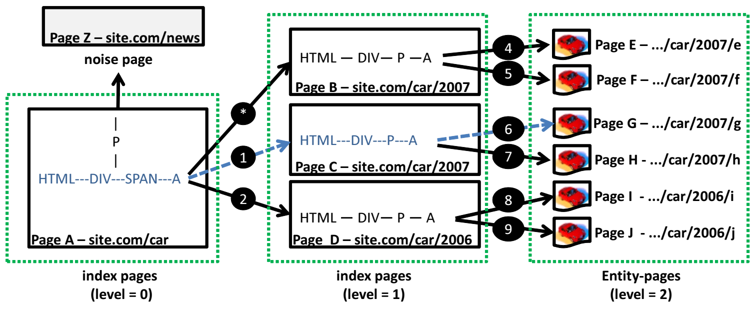


Fig. 1. Entity-tree example. The `Page Z` is highlighed because it is not part of the entity-tree.

root is the index-page with links to all index-pages of the next level (level+1); (5) all pages at a same level have the same degree of topic specificity*; (6) the higher the level of a page the higher its degree of topic specificity; (7) all index-pages at a same level have links to their child nodes in a same link DOM path; and (8) all nodes at the same level of a same entity-tree share a similar URL and HTML structure.*

Example 2. The illustration in Figure 1 describes an entity-tree with height two. The pages that are not part of the logical hierarchy of topics (e.g. `Page Z`) do not belong to the entity-tree and are referred as `noise pages`.

A Web Site can have more than one entity-tree to the same entity-type with a different topic structure. For example, a Web Site with software project entity-pages can be designed in such way that user may navigate through a logical hierarchy of licenses or programming languages. For each type of entity-pages available in a Web site must be at least one entity-tree.

In real Web sites usually occurs a situation that violates the property 5: "*all pages at a same level have the same degree of topic specificity*". This situation, called *pagination*, occurs when an index-page is pointed by an index-page of the same degree of topic specificity instead of an index-page of the immediately preceding degree of topic specificity (more general topic). In this case, we have a pagination page and need a pagination operation.

Definition 4. *(*Pagination page*) Let $p1$ be an index-page, $pp1$ be the parent node of $p1$, the Web page $p2$ is a pagination page if $p1$ and $p2$ have the same degree of topic specificity, $p1$ points to $p2$ and there is not an index-page of the same degree of topic specificity of $pp1$ pointing to $p2$.*

Definition 5. *(*Pagination operation*) Let $p1$ be an index-page, $pp1$ be the parent node of $p1$, and $p2$ be a pagination page pointed by $p1$, then pagination operation states that we need to remove the edge from $p1$ to $p2$ and add an edge from $pp1$ to $p2$.*

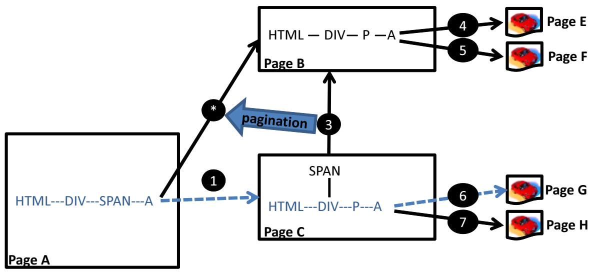


Fig. 2. Example of pagination operation

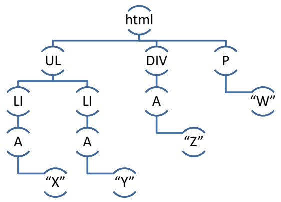


Fig. 3. Example of the DOM tree of a Web page

Example 3. The illustration in Figure 2 shows the pagination operation that generates the valid entity-tree presented in Figure 1. Originally the Web site had the edge 3 and did not have the edge *, violating the property 5. Then, after pagination operation, the edge 3 was removed and the edge * was added.

3.2 Web Page

We consider that the structure of a Web page is defined by its URL and HTML features. The URL feature is defined through a url-schema. A url-schema is the set of terms of the URL.

Definition 6. *(*url-schema*) A url-schema of a Web page p, denoted $v(p)$, is the set of terms of the URL of the p.*

In order to understand our concept of URL term is necessary to know that we see a URL as a sequence of substrings (called *tokens*) split by "/", "?" or "&" characters. Each token is a set of substrings (*sub-tokens*) split by non-alphanumeric characters, changing from letter to digit and vice versa. Then, a URL term is a sub-token associated with the position of the token that contains it. We do not consider the position of each sub-token in a token, since URLs of the Web pages with same entity-type can have different number of sub-tokens.

Definition 7. *(*URL Term*) Let T be a sequence of tokens of a URL u (T_1, T_2, ..., T_n), where T_i occurs in u before T_{i+1}. Each token T_i is a set of sub-tokens ($S_i[1]$, $S_i[2]$, ..., $S_i[n]$), where $S_i[j]$ is the jth sub-token of the token T_i. Each sub-token $S_i[j]$ associated with i is a URL term. An additional URL term is the size of T.*

Example 4. Figure 4 shows an example of a URL, describing all tokens, the sub-tokens of the token T_3 and all URL terms. The additional URL term is highlighted.

The HTML feature is defined through a html-schema. An html-schema is the set of link DOM paths of the HTML.

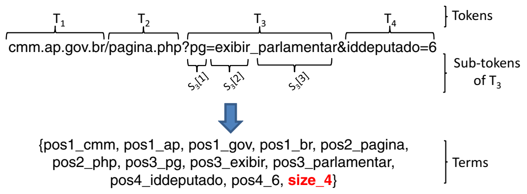

T1
T2
T3
T4
Tokens
cmm.ap.gov.br/pagina.php?pg=exibir_parlamentar&iddeputado=6
S3[1]
S3[2]
S3[3]
Sub-tokens of T3
{pos1_cmm, pos1_ap, pos1_gov, pos1_br, pos2_pagina, pos2_php, pos3_pg, pos3_exibir, pos3_parlamentar, pos4_iddeputado, pos4_6, size_4}
Terms

Fig. 4. Example of tokens, sub-tokens and URL terms

Definition 8. *(*html-schema*) A html-schema of a Web page p, denoted $\Delta(p)$, is the set of link DOM paths in p.*

In order to understand our concept of link DOM path is necessary to know that we see the HTML of Web pages as a Document Object Model (DOM) tree. Then, the link DOM path is a path from the root node to an anchor node.

Definition 9. *(*Link DOM Path*) Let p be a Web page, a link DOM path is a path through the DOM-tree of p that starts from the root and terminates into an anchor node.*

Example 5. The html-schema of the page presented in Figure 3 is $(HTML - UL - LI - A, HTML - DIV - A)$. The path $HTML - P$ does not belong to the html-schema because it does not terminate into an anchor node.

4 SSUP – A URL-Based Method to Entity-Page Discovery

The **SSUP** (**S**tructurally **S**imilar **U**RLs and **P**ages) is a method for entity-page discovery. Specifically, **SSUP** aims to find the set of same type entity-pages in a Web site given a sample entity-page.

We present an overview of our method using the Figure 1. **SSUP** starts with an entity-page as sample page (`Page G`) and find its index-page (`Page C`). Then, a new instance of our problem is recursively triggered using now the index-page found as sample page (finding `Page A` as index-page of the `Page C`). This process is performed until to find the root of the entity-tree (`Page A` in the case). Then, **SSUP** obtains a *sample page path* from root until the given entity-page (`Page A -> Page C -> Page G`). We call *sample page* each page from the sample page path. We call *sample link DOM path* each link DOM path that points to a sample page. The `Page C` is the sample page of the level 1 and "`HTML-DIV-SPAN-A`" is its sample link DOM path.

After, **SSUP** transverses the sample page path from root until the leaf (which is the sample entity-page) catching same level pages. To perform this, in each level x of the entity-tree, **SSUP** catches all pages pointed by the pages of that level and analyzes the structural URL and HTML similarity between these pages and the sample page in the level $x+1$ in order to discard pages that do not belong to the entity-tree. For example, consider Figure 1, analyzing the level 0, **SSUP** catches all pages pointed by `Page A`, including pages `C`, `D` and `Z`. Remember that `Page B` is actually pointed by `Page C`. **SSUP** analyzes the structural URL and HTML similarity between each page with the `Page C` (sample page of level 1) in order to prune noise pages (`Page Z`). By this time, we have pages `C` and `D` in the level 1. Then, **SSUP** needs to discover if these pages contain pagination pages. To perform this, **SSUP** catches all pages pointed by pages `C` and `D` that are structural URL and HTML similar to them (finding the `Page B` as a pagination page of the `Page C`). The process is performed to the next level catching all pages pointed by the pages `B`, `C` and `D`, analyzing the structural URL and HTML similarity between each page with `Page G` (sample page of level 2) and so on.

In the next subsections, we present the similarity metrics used to determine the structural URL and HTML similarity between Web pages (Section 4.1). Finally, we propose four algorithms (Section 4.2). The first algorithm finds the index-page of a given entity/index-page. The second algorithm catches same level pages. The third algorithm catches pagination pages. The last algorithm combines the previous algorithms to perform entity-page discovery.

4.1 Structural Similarity Metrics

In this subsection we present two metrics based on the Web page URL (Weak URL Similarity and Strong URL Similarity) and reuse a metric based on the Web page HTML (sim_{html}) proposed by Blanco et al. [2] in order to measure the structural similarity between two Web pages. The similarity metrics Weak URL Similarity and HTML Similarity are used to determine if a page is a pagination of other. The similarity metrics Strong URL Similarity and HTML Similarity are used to determine if two pages are in the same level in the entity-tree.

Weak URL Similarity is a simple similarity metric that compares two Web pages based on the terms that belong to their `url-schema`. This metric gives the same importance to each term. So, if two URLs share the term "`www`" or the term "`driver`" has the same impact in the result.

Definition 10. (Weak URL Similarity) *Let $v(p_1)$ be the url-schema of the Web page p_1 and $v(p_2)$ be the url-schema of the Web page p_2, the weak URL similarity between p_1 and p_2 is defined as:*

$$weaksim_{url}(p_1, p_2) = \frac{|v(p_1) \cap v(p_2)|}{|v(p_1) \cup v(p_2)|} \tag{1}$$

The Strong URL Similarity also compares two Web pages based on the terms that belong to their `url-schema`. However, it is more robust than Weak URL similarity since it assigns a different weight to each URL term according to its importance to distinguish same level pages from non-same level pages. In this metric, the URL terms are assumed to be all mutually independent. The URL of the sample page is represented as vector of URL term weights in a n-dimensional space, in which n is the total number of URL terms.

Definition 11. *(*Strong URL Similarity*) Let $v(sp)$ be the url-schema of the sample page sp, $v(p)$ be the url-schema of the Web page p, which we desire to compare with sp, and ip be the index-page of sp, the strong URL similarity between sp and p is defined as:*

$$strongsim_{url}(ip, sp, p) = \frac{\sum_{t \in v(sp) \cap v(p)} W_t(ip, sp)}{\sum_{t \in v(sp)} W_t(ip, sp)} \tag{2}$$

The weight of a URL term used in the Strong URL Similarity is based on the observation of Weninger et al. [8] "lists usually contain items which are similar in type or in content" and our observation "entity-pages of the same type usually share a common URL structure". In our context, we have a sample page and we want to know how are other web pages similar to sample page based on their URLs. The idea is that terms that occur in many URLs, from links of the index-pages, in the same link DOM path that the sample page, and occur in few link DOM paths from the index-pages are more important because they have a high discriminating power. For example, in Table 1, considering `L1` as sample page and $P1$ as its index-page, the term "`pos2_car`" should have the highest weight because it occurs in five URLs that are in the same link DOM path of the sample page ("`HTML-UL-LI-A`") and does not occur in other link DOM paths.

Definition 12. *(*URL Term Weight*) Let ip be the index-page of the sample page sp. Let P be a set of link DOM paths (P_1, P_2, ..., P_n) from ip. Let $P_i = (P_i[1], P_i[2], ..., P_i[n])$, where $P_i[j]$ is the url-schema of the jth URL in the link DOM path P_i. Let P_{sp} be the P_i that contains $v(sp)$, and t a URL term from $v(sp)$, the weight of t from sp in ip is defined as:*

$$W_t(ip, sp) = TF_{t,P_{sp}} \times IDF_{t,P} \tag{3}$$

where, $TF_{t,P_{sp}}$ is the number of url-schemas in P_{sp} that contains the term t and $IDF_{t,P}$ is defined as:

$$IDF_{t,P}(ip) = log(\frac{|P|}{|P_i \in P : t \in P_i|})$$

Example 6. The Table 1 presents an example of same level index-pages with their links and the link DOM path that points to each link. For each link, we show its strong URL similarity with the sample page $L1$. The highest strong URL similarity is with itself (`1.00`), followed by pages `L2` and `L3` that share with $L1$ the sub-token "`car`" in the second token and the sub-token "`ford`" in the third token. These sub-tokens occur only in the link DOM path of the sample page. The strong URL similarity of the `Page L7` is 0 because all sub-tokens shared between this page and the sample page (e.g., "www") occurs in all link DOM paths and all occurrences are in the first token. Pages L0 and L8 do not have ss_u because they are in a link DOM path different from sample page.

HTML Similarity is a simple similarity metric that compares two Web pages based on the link DOM paths that belong to their `html-schema`. This metric gives the same importance to each link DOM Path.

Table 1. An example of same level index-pages with their link DOM paths and the pages pointed by each link DOM path (column `Links`). The column ss_u shows the strong URL similarity between each page in the column `Links` with the sample page `L1`.

Same Level Index-pages	Link DOM Path	Links	ss_u
P1	HTML-DIV-A	L0 = www.website.com/privacy_policy.htm	-
	HTML-UL-LI-A	L1 = www.website.com/car/ford/ba-falcon-rx-8	1.00
		L2 = www.website.com/car/ford/svt-mustang-cobra-coupe	0.68
		L3 = www.website.com/car/ford/mustang-mach-1	0.68
P2		L4 = www.website.com/car/ferrari/360-challenge-stradale	0.53
P3		L5 = www.website.com/car/audi/a4-cabriolet-3.0	0.53
P4		L6 = www.website.com/test_drive/rx-8.htm	0.21
P5		L7 = www.website.com/news.htm	0.00
	HTML-DIV-A	L8 =www.website.com/wallpapers.htm	-

Definition 13. *(*HTML Similarity*) Let $\Delta(p_1)$ be the html-schema of the Web page p_1 and $\Delta(p_2)$ be the html-schema of the Web page p_2, the HTML similarity between p_1 and p_2 is defined as:*

$$sim_{html}(p_1, p_2) = \frac{|\Delta(p_1) \cap \Delta(p_2)|}{|\Delta(p_1) \cup \Delta(p_2)|} \tag{4}$$

4.2 Algorithms

In this subsection we present the algorithms that compose the method **SSUP**. The Algorithm 1 aims to find the index-page of a given page. The intuition behind the algorithm is that the index-page of a given page *sp* is the page delivering the largest number of more structurally similar URLs to *sp*. The solution presented by the algorithm collects the pages pointed by the given page (line 4). Then, the pages that do not have a link to the given page are removed (line 5). Finally, the algorithm returns the page with the max candidate index-page weight (line 6).

The candidate index-page weight accumulates the Strong URL Similarity between its links and a given page. It is considered only the links in the same link DOM path that the given page. The Algorithm 1 uses this weight to find the index-page of a given page, choosing the candidate index-page with the highest weight.

Algorithm 1. Top Index-Page algorithm

```
1: INPUT: a sample page $sp$;
2: OUTPUT: the top index-page;
3: begin $top\_index\_page(sp)$
4: $P$ = GetLinks($sp$);
5: remove $p_i$ from $P$ where $p_i$ does not have a link to $sp$;
6: return $p_i$ from $P$ with max $W_{ci}(p_i, sp)$; //Equation 5
7: end
```

Algorithm 2. Catching Same Level Pages algorithm

```
1: INPUT: a set of index-pages of the level $x$ ($IP$), the sample page of the level $x+1$ ($sp$), the sample link DOM path of $sp$ ($sdp$);
2: OUTPUT: the set of pages of the level $x+1$ pointed by $IP$;
3: begin catching_same_level_pages($IP, sp, sdp$)
4: add $sp$ to $rs$;
5: for each $ip_i$ IN $IP$ do
6:       add GetLinksInPath($ip_i$, $sdp$) to $L$;
7: end for
8: create a list of groups $G = (G_1, G_2, ..., G_n)$, where each group $G_i$ contains each link $l_i \in L$ (except $sp$) with the same $strongsim_{url}$ with $sp$;
9: sort $G$ by $strongsim_{url}$ decreasing order;
10: add all links from $G_1$ to $rs$;
11: $minsim_{html} = \min_{l_i \in G_1} sim_{html}(sp, l_i)$;
12: remove $G_1$ from $G$
13: for each $G_i$ IN $G$ do
14:       if $\max_{l_i \in G_i} sim_{html}(sp, l_i) >= minsim_{html}$ then
15:             add all links from $G_i$ to $rs$;
16:       else
17:             break;
18:       end if
19: end for
20: return $rs$
21: end
```

Definition 14. *(*Candidate index-page weight*) Let p be a candidate index-page for the Web page sp, L be a set of pages pointed by p through the same link DOM path that sp, the candidate index-page weight of p for sp is defined as:*

$$W_{ci}(p, sp) = \sum_{l \in L} strongsim_{url}(p, sp, l) \quad (5)$$

The goal of the Algorithm 2 is given a set of index-pages of level x and the sample page of the level $x+1$, finds all pages of level $x+1$ delivered by the index-pages. The intuition behind the algorithm is that same level pages have structurally similar URLs and HTMLs. The solution proposed is that same level pages have larger $strongsim_{url}$ and sim_{html} than non-same level pages.

The Figure 5 presents an example of the execution of the Algorithm 2, considering the pages `P1`, `P2`, `P3`, `P4` and `P5` as index-pages of level x (called just index-pages); `L1` as the sample page of level $x+1$ (called just sample page) and "`HTML-UL-LI-A`" as the sample link DOM path of the sample page, i.e., the path in the index-page that points to sample page. The Table 1 describes these pages. The goal is to return all pages of level $x+1$. In the line 4, the sample page is added to the result set. In lines 5-9, the Web pages pointed by the index-pages through the sample link DOM path, except sample page, are collected and grouped by their strong URL similarity with the sample page. The

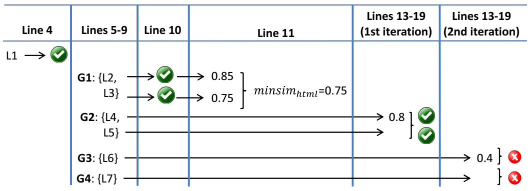


Fig. 5. An execution of the Algorithm 2. Input: IP=(`P1`, `P2`, `P3`, `P4`, `P5`), SP=`L1`, sdp=`HTML-UL-LI-A` (described in Table 1). Ok icon means that the Web page was added to the result set. Not ok icon means that the Web page was discarded.

groups are sorted by their strong URL similarity decreasing order. Four groups are created: `G1`, `G2`, `G3`, `G4`. The group `G1` (`G4`) has the Web pages with largest (lowest) strong URL similarity value with the sample page. In line 10, all Web pages from `G1` (`L2` and `L3`) are added to result set. In line 11, we compute the min HTML similarity between the sample page and the pages from `G1`, called $minsim_{html}$. In this example, the $minsim_{html}$ is the HTML similarity between sample page and `L3` (0.75). In line 12, omitted in the figure, the `G1` is removed from the list of groups. The first iteration of the "*for*" (lines 13-19) analyzes the group `G2`, since it has the largest strong URL similarity after `G1` has been removed. It is computed the HTML similarity between the first Web page of the `G2` (`L4`) and the sample page. As the HTML similarity value is greater than $minsim_{html}$ all Web pages from `G2` are added to result set, without compute the HTML similarity for other Web pages of `G2`. The second iteration of the "*for*" (lines 13-19) analyzes the group `G3`. Since the min HTML similarity between sample page and the pages of `G3` is less than $minsim_{html}$, so all pages of the group (`L6`) are discarded and all remain groups (`G4`) are discarded too. This follows the idea "the same level pages have larger $strongsim_{url}$ and sim_{html} than non-same level pages". Finally, the result set (with pages `L1`, `L2`, `L3`, `L4` and `L5`) is returned in line 20.

The Algorithm 2 fails in catch pagination pages because these pages are not pointed by index-pages of the preceding level, but by pages of the same level. So, the Algorithm 3 aims to catch these pages. The intuition behind this algorithm is that pages delivered by index-pages of the level x with HTML similarity to pages of the level x larger than to pages of the level $x+1$ are strong candidate to be pagination pages. The solution presented in the algorithm receives as input: (i) a set of index-pages of the level x; (ii) a sample page of the level x (sp_x); and (iii) a sample page of the level $x+1$ (sp_{x+1}). Then, in lines 4-6, the pages pointed by the index-pages are collected. These pages are grouped according to their Weak URL Similarity with sp_{x+1} (line 7). After, one page of each group has its HTML similarity computed with sp_x and with sp_{x+1}, and if the first score is greater or equal to the second score, then all pages of the group are added to the result set (lines 8-13). Finally, the result set is returned (line 14).

It is important to note that this step catches noise pages. But these pages are pruned when a next level is analyzed because noise pages do not point to pages with structurally similar URL and HTML to the sample page.

Algorithm 3. Pagination algorithm

```
1: INPUT: the sample page of the level $x$ ($sp_x$), the sample page of the level $x+1$ ($sp_{x+1}$), index-pages of the level $x$ ($IP$);
2: OUTPUT: a set of pagination pages of the level $x$ pointed by the $IP$;
3: begin pagination($sp_x$, $sp_{x+1}$, $IP$)
4: for each $ip_i$ IN $IP$ do
5:     add GetLinks($ip_i$) to $L$;
6: end for
7: create a list of groups $G = (G_1, G_2, ..., G_n)$, where each group $G_i$ contains each link $l_i \in L$ with the same $weaksim_{url}(sp_{x+1}, l_i)$ and has the same link DOM path;
8: for each $G_i$ IN $G$ do
9:     $l_0$ = first link from $G_i$
10:    if $sim_{html}(sp_x, l_0) >= sim_{html}(sp_{x+1}, l_0)$ then
11:        add all links from $G_i$ to $rs$;
12:    end if
13: end for
14: return $rs$
15: end
```

The Algorithm 4 shows the overall SSUP algorithm. In lines 4-12 the sample page path is created by calling recursively the function *top_index_page*. Note that we do not have an approach to discovery the root node, instead we have a parameter that estimates the height of the entity-tree. If this parameter is less than the real entity-tree height then some entity-pages will not be discovered. If this parameter is greater than the real entity-tree height then some unnecessary pages will be downloaded, but the quality results is not affected since the noise pages are pruned when we analyze the URL and HTML structural similarity. However, we add an additional stopping criterion (lines 7-9): when an index-page discovered for a level x was already discovered for a level y where $y > x$. In lines 13-22, we transverse the entity-tree from root to leaves catching same level entity-pages through the functions *catching_same_level_pages* and *pagination*. It is important to note in lines 19-21 that entity-pages do not have pagination.

5 Experiments

In this section, we describe the experiments we performed in order to evaluate the effectiveness of **SSUP**. We compared **SSUP** with Indesit [2] and GPP [8]. We chose to compare our method with these baselines among the methods that focus on surface Web because they depend neither on the specific HTML markup [7] nor on a human-compiled encyclopedia [6].

In order to analyze the results of experiments, we evaluated them considering the following measures: (i) recall, (ii) precision; (iii) F1-measure; (iv) number of downloaded pages; (v) processing time and (vi) T-test. T-test is a statistical hypothesis test, and recall, precision and F1-measure are well known quality measure metrics in IR community [1]. We show just F1-measure results in terms

Algorithm 4. SSUP algorithm

```
1: INPUT: one sample entity-page $sp$, the entity-tree height $h$;
2: OUTPUT: the set of entity-pages of the same type that $sp$ contained in the Web site;

3: begin ssup(sp, r)
4: add $sp$ to $SPP[h]$; //SPP = Sample Page Path
5: for $i$=0 until $h - 1$ do
6:       $tmp = top\_index\_page(SPP[h - i])$;
7:       if $SPP$ contains $tmp$ then
8:             break;
9:       end if
10:      $aux = h - i - 1$;
11:      add $tmp$ to $SSP[aux]$;
12: end for
13: for $i$=$aux$ until $h - 1$ do
14:      if $i = aux$ then
15:            add $SPP[i]$ to $IP$;
16:      end if
17:      $sdp$ = the link DOM path in $SPP[i]$ that contains the link to $SPP[i + 1]$;
18:      add $catching\_same\_level\_pages(IP, SPP[i + 1], sdp)$ to $IP$;
19:      if $i < h$ - 2 then
20:            add $pagination(SPP[i + 1], SPP[i + 2], IP)$ to $IP$;
21:      end if
22: end for
23: return $IP$
24: end
```

of quality because recall and precision are combined in F1-measure. In the efficiency aspect we evaluated the total number of downloaded pages and the processing time. We previously downloaded the pages of each Web site used in order to avoid the network interference in the processing time.

5.1 Setup

The experiments were performed on a Intel Core 2 Quad 2.66GHz running Ubuntu 9.10, with 8GB of main memory and 5TB of disk space. **SSUP**, `Indesit` and GPP were implemented in Java. We created 28 datasets, each one from a specific Web site, considering a specific entity-type. We created two groups with these datasets: (i) `multiple type group` with 10 datasets of different entity-types (except `association` that repeats once), described in Table 2; and (ii) `council type group` with 18 datasets of council member entity-type. In the last group, we analyzed the official Web sites of the council of the 26 Brazilian state capitals. However, we excluded: 4 Web sites because they do not have an entity-page for each council member; 3 Web sites because the council member entity-pages are internal frames of the index-page; and 1 Web site because it does not allow crawling its pages.

Table 2. Datasets of the multiple type group

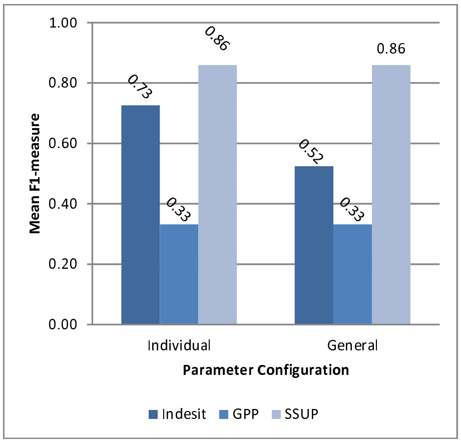
ID	Web site	Entity-type	NEP
Fifa	www.fifa.com	Association	209
GP2	www.gp2series.com	Driver	32
Guitar	www.guitaretab.com	Group	32318
MIT EECS	www.eecs.mit.edu	People	1042
Olympic-A	www.olympic.org	Association	204
Olympic-S	www.olympic.org	Sport	56
Pgfoundry	pgfoundry.org	Software Project	382
Senado	www.senado.gov.br	Senator	121
Stanford EE	engineering.stanford.edu	Staff	473
Supercar	www.supercarsite.net	Car	512

Label: NEP: number of entity-pages

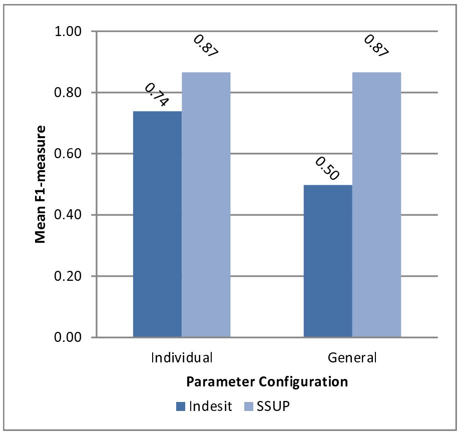
`Indesit` has two parameters: (i) `T-value` - the html-schema similarity threshold between two Web pages varying from 0 to 1; (ii) `Tag features` - defines if the `Indesit` considers as a link DOM path only tags; tags and attribute names; or tags, attribute names and attribute values. We tested all combinations between T-values (0.1; 0.2; ...; 0.9) and tag features. The best parameter configuration related to quality of results in the `multiple type group` was `T-value`$= 0.5$ and `Tag feature` = tags and attribute names; and in the `council type group` was `T-value`$= 0.8$ and `Tag feature` = tags and attribute names. **SSUP** has one parameter: `H-value` - the estimate of the entity-tree height. We tested the `H-values`: 1, 2, ..., 5. The best `H-value` related to quality of results was 2 and 1 in the `multiple type group` and `council type group`, respectively. `GPP` has one parameter: `K-value` - the number of iterations. We tested only one iteration because the processing time of `GPP` is too high compared with `Indesit` and `SSUP`. We ran all methods with each configuration three times, each one with a different sample entity-page in order to verify that the behavior of the method does not depend on specific features of the sample entity-page. The three sample entity-pages were chosen randomly from the first page displayed (when there is pagination).

5.2 Comparison

The goal of these experiments is to compare the methods **SSUP**, `Indesit` and `GPP` in terms of quality and efficiency. For **SSUP** and `Indesit` we show the results with the best parameter configuration in terms of quality results in each individual dataset (`individual` configuration) and with the best parameter configuration in terms of quality results considering all datasets of the group analyzed (`general` configuration). We run `GPP` with only one iteration (so, individual and general configuration are the same one) and only on the `multiple type group` because its processing time is too high.



(a) `multiple type group`



(b) `council type group`

Fig. 6. Mean F1-measure comparison

Figure 6 shows a comparison between the F1-measure of **SSUP**, `Indesit` and `GPP` on the `multiple type group` (Figure 6a) and between **SSUP** and `Indesit` on the `council type group` (Figure 6b). SSUP outperformed `Indesit` and `GPP`. There is a statistical significant difference (p-value < 0.05) between the F1-measure values of the methods (except between `Indesit` and `SSUP` with individual configuration on the `multiple type group` where p-value was 0.50126). Considering the general configuration, **SSUP** had better quality results than `Indesit` because it learns the threshold of html-schema similarity in each dataset from the groups of Web pages with high structurally URL similarity with the sample entity-page and that are in the same link DOM path that the sample entity-page. On the other hand, in `Indesit` the user should define a threshold of html-schema similarity between entity-pages and this threshold varies from dataset to dataset. Considering the individual configuration (the smallest difference in terms of F1-measure between **SSUP** and `Indesit`), **SSUP** only outperformed `Indesit` in datasets that contain an intersection of entity-pages and non-entity pages related to the html-schema similarity. In these cases, in `Indesit` if we increase the T-value the precision increases, but the recall decreases. If we decrease the T-value the recall increases, but the precision decreases because the method catches erroneous pages. On the other hand, **SSUP** uses the structurally URL similarity as an additional feature to discriminate entity-pages from non entity-pages. `GPP` presented the worst values of F1-measure because, in many cases, one iteration was not enough to find most entity-pages. However, even on the `Olympic - S` dataset where `GPP` found all entity-pages (recall = 1) many erroneous pages were returned decreasing the precision to 0.32 because the host city entity-pages have similar HTML and visual features to sport entity-pages.

The main SSUP fail cases occurred when: (i) the entity-pages do not contain a link to their index-page affecting the recall of SSUP; (ii) some entity-pages have different link DOM paths affecting the recall of SSUP; (iii) the URL of the

Table 3. Average of downloaded pages and processing time on the `multiple type group`

	Indesit Individual	**Indesit General**	**GPP**	**SSUP Individual**	**SSUP General**
Downloaded Pages	1680	980	212	778	781
Processing Time	854	779	2984	46	47

entity-pages and non entity-pages are composed of the domain plus "/" plus one token affecting the precision.

The average of downloaded pages on the `multiple type group` for each method is presented in Table 3. Considering individual configuration, the mean number of downloaded pages of `Indesit` was 1680 while **SSUP** was 778 (53.69% less than `Indesit` and p-value < 0.05). This occurred because **SSUP** filters out some pages just by their URLs, without needing to download them while `Indesit` needs to download a Web page to evaluate it. Considering general configuration, the mean number of downloaded pages of `Indesit` was 980 while `SSUP` was 781 (20.31% less than `Indesit`). However, this difference is not statistically significant (p-value > 0.05). This occurred because `Indesit` with general configuration had a low recall in many datasets, consecutively it downloaded fewer pages. `GPP` presented the lowest number of downloaded pages (72.86% less than **SSUP** with general configuration and p-value < 0.05). The main reason of this big difference is the low recall in most datasets. However, even in the `Olympic - S` dataset where all entity-pages were found, the number of downloaded pages was lower. This occurred because we used only one iteration, but if we increase the number of iterations to increase the recall, more pages are downloaded.

The average of processing time for **SSUP**, `Indesit` and `GPP` on the `multiple type group` is presented in Table 3. **SSUP** had the shortest processing time (with p-value < 0.05 in all cases). This behavior can be explained because **SSUP** filters out some pages just by their URLs, without needing to analyze the HTML of them while `Indesit` needs to analyze the HTML of all downloaded pages. Moreover, after computing the Sample Page Path, **SSUP** analyzes in each Web page only the sample link DOM path while `Indesit` analyzes all link DOM paths in all Web pages downloaded. `GPP` had the worst values of processing time, even having downloaded less Web pages, because it explores visual information and to perform this it is necessary to render the Web page in a browser loading all images, css and javascripts, which greatly increases the processing time.

The experiments showed that **SSUP** outperforms `Indesit` related to the quality results when we define one unique parameter configuration for all datasets. When we use the best parameter configuration for each individual dataset, **SSUP** improves the quality results when the dataset has an intersection between entity-pages and non-entity pages related to the html-schema similarity. In other cases, **SSUP** does not improve the quality results, but it reduces the number of downloaded pages and processing time. Furthermore, the **SSUP** parameter - H - can be defined by a user visually navigating through the Web site while the `Indesit` parameters (T and tag feature) require a processing of the entity-pages. Then, we can conclude that combining URL and HTML features improves the quality results of entity-page discovery and decreases the number of downloaded pages and the processing time.

6 Conclusions

In this paper we propose a new method to entity-page discovery, called **SSUP**. Specifically, given a sample entity page of a Web site, we want to find the set of same type entity-pages of that Web site. The main contribution of **SSUP** is to combine URL and HTML features to entity-page discovery. We first define the basic structures of a Web site and a Web page according to our method. Second, we propose two structural URL similarity metrics and reuse a HTML similarity metric. Thirdly, we develop a set of algorithms that compose the method. We demonstrated the effectiveness of **SSUP** by comparing it with `Indesit` and `GPP` using real datasets.

The experiments have showed that combining URL and HTML features improves the results in terms of quality, number of downloaded pages and processing time. Entity-pages dynamically generated by populating fixed pages templates with content from a back-end DBMS have the better results. However, we are working on incorporating other features besides URL and HTML to achieve good results in people entity-pages created by different users, since these pages usually do not contain a link to its index-page affecting the recall of SSUP. We are also creating a parallelized version of SSUP.

References

1. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval: The Concepts and Technology Behind Search. Addison Wesley Professional (2011)
2. Blanco, L., Crescenzi, V., Merialdo, P.: Efficiently locating collections of web pages to wrap. In: WEBIST, pp. 247–254. INSTICC Press (2005)
3. Blanco, L., Dalvi, N.N., Machanavajjhala, A.: Highly efficient algorithms for structural clustering of large websites. In: WWW, pp. 437–446. ACM (2011)
4. Grünwald, P.D.: The Minimum Description Length Principle (Adaptive Computation and Machine Learning). The MIT Press (2007)
5. He, Y., Xin, D., Ganti, V., Rajaraman, S., Shah, N.: Crawling deep web entity pages. In: WSDM, pp. 355–364. ACM (2013)
6. Kaptein, R., Serdyukov, P., de Vries, A.P., Kamps, J.: Entity ranking using wikipedia as a pivot. In: CIKM, pp. 69–78. ACM (2010)
7. Lerman, K., Getoor, L., Minton, S., Knoblock, C.A.: Using the structure of web sites for automatic segmentation of tables. In: SIGMOD Conf., pp. 119–130. ACM (2004)
8. Weninger, T., Johnston, T.J., Han, J.: The parallel path framework for entity discovery on the web. ACM Trans. Web 7(3), 16:1–16:29 (2013)
9. Yu, H., Han, J., Chang, K.C.C.: Pebl: Web page classification without negative examples. IEEE Trans. on Knowl. and Data Eng. 16(1), 70–81 (2004)

StreamMyRelevance!
Prediction of Result Relevance from Real-Time Interactions and Its Application to Hotel Search

Maximilian Speicher[1,2], Sebastian Nuck[2,3], Andreas Both[2], and Martin Gaedke[1]

[1] Chemnitz University of Technology, 09111 Chemnitz, Germany
[2] R&D, Unister GmbH, 04109 Leipzig, Germany
[3] Leipzig University of Applied Sciences, 04277 Leipzig, Germany
maximilian.speicher@s2013.tu-chemnitz.de,
martin.gaedke@informatik.tu-chemnitz.de,
andreas.both@unister.de, sebnuck@gmail.com

Abstract. The prime aspect of quality for search-driven web applications is to provide users with the best possible results for a given query. Thus, it is necessary to predict the relevance of results *a priori*. Current solutions mostly engage clicks on results for respective predictions, but research has shown that it is highly beneficial to also consider additional features of user interaction. Nowadays, such interactions are produced in steadily growing amounts by internet users. Processing these amounts calls for streaming-based approaches and incrementally updateable relevance models. We present *StreamMyRelevance!*—a novel streaming-based system for ensuring quality of ranking in search engines. Our approach provides a complete pipeline from collecting interactions in real-time to processing them incrementally on the server side. We conducted a large-scale evaluation with real-world data from the hotel search domain. Results show that our system yields predictions as good as those of competing state-of-the-art systems, but by design of the underlying framework at higher efficiency, robustness, and scalability.

Keywords: Streaming, Real-Time, Interaction Tracking, Learning to Rank, Relevance Prediction.

1 Introduction

Nowadays, search engines are among the most important and most popular web applications. They are essential for supporting users with finding specific pieces of information on the web. Thus, their *prime aspect of quality* is to ensure that relevant results are displayed where they receive the highest attention. In other words, the ranking of results is a major quality aspect in the context of the search application as a whole. This makes it necessary to estimate the relevance of results *a priori*. Common methods for obtaining such estimates are generative *click models* (e.g., [3,4,15]). Based on certain assumptions about user behavior, these models predict the relevance of a certain result taking into account the number of clicks it has received for a given query. However, click data are not a perfect indicator concerning relevance since users might return to the search engine results page (SERP) after having clicked a useless result. Additionally, search engines more and more try to answer queries directly on the SERP, e.g., as Google do with their *Knowledge Graph*[1]. Thus, additional information that complement click data should be taken into account for predicting relevance, e.g., in terms of dwell times on landing pages [9] or other client-side user behavior (e.g., [9,13,18]). Previous research has shown the value of such page-level interactions [11,13,20]. Also, generative [12] as well as discriminative [20] approaches to relevance prediction exist that engage user behavior other than clicks only.

With a growing amount of users, it is possible for search engine providers to collect enormous amounts of client-side data. This is particularly the case if we consider interactions other than clicks. Along with the increasing quantities of tracking data, a short time-to-market becomes more and more important. That is, providers need to quickly analyze collected information and feed potential findings back into their products to ensure user satisfaction. This calls for the use of novel systems for data stream mining, such as *Storm*[2], which are currently gaining popularity in industry and research. These systems can help to cope with the seemingly endless streams of data produced by today's internet users. Yet, none of the approaches for relevance prediction mentioned above leverages data stream mining to process collected information.

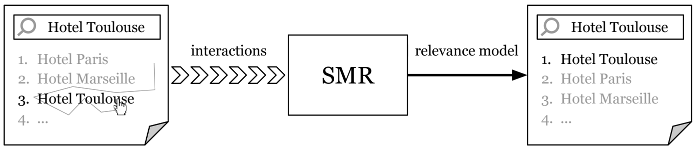


Fig. 1. The intention behind StreamMyRelevance!—from collecting a stream of user interactions to reordering search results based on relevance models

We present *StreamMyRelevance!* (SMR), which is a novel streaming-based system for ensuring ranking quality in search engines. Our system caters for the whole process from tracking interactions to learning incremental *relevance models*, i.e., models that predict the relevance of a search result (for a given query) based on certain features of user interaction. The latter can be used to directly feed predictions back into the ranking process of the search engine, e.g., as a weighted factor in a learning-to-rank function (cf. Fig. 1). SMR is based on Storm and leverages tracking and data processing functionalities provided by *TellMyRelevance!* (TMR)—a pipeline that has proven its effectiveness in predicting search result relevance [20]. Yet, TMR is a batch-oriented approach that does not provide means for incrementally learning relevance models on

[1] `http://www.google.com/insidesearch/features/search/knowledge.html` (2013-09-06).
[2] `http://www.storm-project.net/` (2013-12-30).

a streaming basis. Thus, SMR wraps the borrowed functionalities into a new system that is able to handle real-time streams. Our system has three main advantages over existing approaches, i.e., (1) considering interactions other than clicks for predicting relevance, (2) collecting and processing these interactions as a stream and (3) providing incremental relevance models that do not require re-processing of previously processed data. Based on this, the main hypothesis investigated in this paper is as follows: *SMR is able to achieve the same relevance prediction quality as TMR at better efficiency, robustness and scalability.*

We have evaluated SMR in terms of its feasibility and quality of relevance predictions. For this, large amounts of real-world data from two *hotel booking portals* were available. A comparison to TMR has been performed, which due to its batch-oriented design has look-ahead capabilities and thus more information available [20]. Still, our results show that SMR's prediction quality is not significantly worse compared to TMR. Moreover, our system in parts compares favorably with predictions of the Bayesian Browsing Model (BBM) [16], a *state-of-the-art* generative click model successfully applied in industry. Furthermore, reviews of efficiency, robustness and scalability show that SMR compares favorably with the competing approaches in these respects.

In the following section, we describe important concepts our work is based on, before giving an overview of related work. Section 3 explains the design and architecture of SMR, followed by an evaluation of effectiveness, efficiency, robustness and scalability of SMR and competing approaches in Section 4. Limitations and potential future work are addressed in Section 5, before giving concluding remarks in Section 6.

2 Background and Related Work

The following gives background information on the underlying concepts of Storm [17], which are important for understanding the architecture of SMR.

The logic of a Storm application is represented as a graph consisting of **spouts** and **bolts** that are connected by *streams*, i.e., unbounded sequences of data tuples. This concept is called a **topology**. On the one hand, spouts act as sources of streams by reading from external data sources (e.g., a DB) and emitting tuples into the topology. On the other hand, bolts are the core processing units of a topology. They receive tuples, process the contained data and emit results as a new stream. Spouts and bolts can have multiple outgoing streams, which provides the possibility of separating tuples within bolts and emitting them using different streams.

The direct competitor to Storm is Yahoo!'s S4[3]. It as well provides distributed stream computing functionality, but its underlying concepts and configuration are more complex[4]. As described in [22], benchmarks have shown that S4 is almost 10 times slower than Storm.

This research is related to a variety of existing work in the fields of *relevance prediction* and *data stream mining*. An overview will be given in the following.

[3] `http://incubator.apache.org/s4/` (2013-09-28).
[4] `http://demeter.inf.ed.ac.uk/cross/docs/s4vStorm.pdf` (2014-01-06).

Concerning the relevance of search results, it is necessary to rely on human relevance judgments—i.e., asking the user to explicitly rate the relevance of a result—for the best possible predictions. However, since such data are usually not available in large numbers, different solutions are required. Joachims [15] proposes to use *clickthrough data* instead of human relevance judgments. Based on the *cascade hypothesis* [4,16], i.e., the user examines results top-down and neglects results below the first click, it is possible to infer relative relevances. That is, the clicked result is more relevant than the non-clicked results at higher positions. Using such relative relevances, Joachims engages clickthrough data as training data for learning retrieval functions with a support vector machine approach [15]. In contrast to the above, models like the *Dependent Click Model* [8] assume that more than one result can receive clicks. That is, results below a clicked position might be examined and thus also clicked if they are relevant.

The *Dynamic Bayesian Network Click Model* (DBN) described in [3] generalizes the *Cascade Model* [4] by aiming at relevance predictions that are not influenced by position bias. To achieve this, the authors (besides the perceived relevance of a search result) also consider users' satisfaction with the website linked by the clicked result.

Generally, click models are based on the *examination hypothesis*, which states that only relevant search results that have been examined are clicked [16]. Yet, not all of these models follow the cascade hypothesis. All of the above described are *generative* click models that try to provide an alternative to explicit human judgments by *predicting the relevance of search results* based on click logs. The main differences to SMR are that we aim at predicting relevance using a *discriminative* approach also taking into account *interactions other than clicks*. Moreover, the above click models are not designed for efficient processing of *massive data streams* or *incremental updates*.

The *Bayesian Browsing Model* (BBM) [16] is based on the *User Browsing Model* (UBM) [7], which assumes that the probability of examination depends on the position of the last click and the distance to the current result [16]. Contrary to UBM, BBM aims at scalability to petabyte-scale data and incremental updates. The authors compute "relevance posterior[s] in closed form after a single pass over the log data" [16]. This enables incremental learning of the click model while making iterations unnecessary. Still, contrary to SMR, BBM is again a *generative* model that does not leverage the advantages of additional interaction data.

Concerning user interactions other than clicks, in [11], Huang has found that these are a valuable source of information for relevance prediction. Following, Huang et al. [13] investigate the correlations between human relevance judgments and mouse features such as hover time and unclicked hovers, among others. They find positive correlationsand conclude that these can be used for inferring search result relevance. Also, part of our system is based on a scalable approach for collecting client-side interactions described by the authors [13].

In [9], Guo and Agichtein present their *Post-Click Behavior Model*. They incorporate interactions like cursor or scrolling speed on a landing page into determining its relevance, i.e., interactions that happen *post-click*. This is also partly related to DBN [3], where the relevance of the landing page is modeled separately from the perceived relevance of the result. While this approach is promising for inferring the actual usefulness of a landing page, it would be difficult to realize since search engines would need access to landing page interactions through, e.g., a browser plug-in or tracking scripts.

Making use of scrolling and hover interactions, Huang et al. [12] extend the Dynamic Bayesian Network Click Model described earlier to leverage information beyond click logs. Their results show that this improves the performance in terms of predicting future clicks compared to the baseline model. While this *generative* approach involves interactions other than clicks, in contrast to SMR, it does not specifically aim at incremental learning or efficient processing of massive data streams.

TMR is a system described by Speicher et al. [20]. Parts of SMR are based on this work, particularly in terms of client-side interaction tracking, preprocessing of raw data and computation of interaction features. Like SMR, TMR is a *discriminative* approach to relevance prediction, but in contrast is a batch-oriented system. In particular, its relevance models are not trained incrementally, i.e., *all* data have to be re-processed before obtaining an updated model.

3 SMR: Streaming Interaction Data for Learning Relevance Models

The following Section describes SMR, which is organized as a streaming-based process. Its aim is to enable processing of big data streams while leveraging the advantages of user interaction data for the prediction of search result relevance. This supports more optimal ranking of results, which is a major quality aspect of search-driven web applications.

The system comprises four main components as illustrated in Fig. 2: The **Client-Side Interaction Tracking** component in terms of a jQuery plug-in; The **Preprocessor** for reading and preprocessing streams of tracking data and

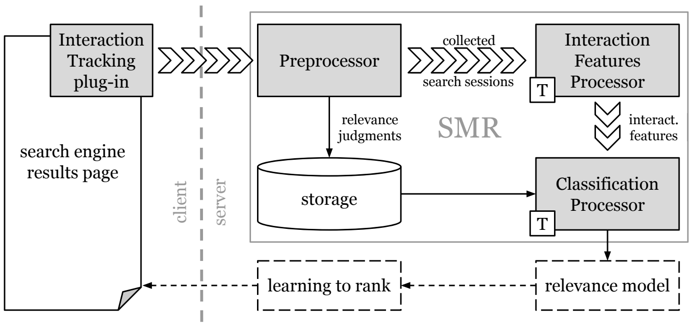


Fig. 2. The main components and process flow of SMR (Streams are visualized by sequences of chevrons; Storm topologies are annotated using a "T")

relevance judgments; The **Interaction Features Processor** for calculating interaction features from tracking data; The **Classification Processor** for incrementally training a relevance model using the previously computed features and collected relevance judgments.

Our Storm-based system has been specifically designed with an incremental approach in mind. The four steps above can be regarded as a sequence of independent processes. That is, the results of each step as well as the resulting relevance models are persisted (temporarily). As a result, in case of a crash within the system, SMR can resume its work at the step prior to the incident without starting over from the very beginning.

3.1 Client-Side Interaction Tracking

For client-side interaction tracking, SMR builds upon a "minimally invasive jQuery plug-in" [20] that is provided by TMR. This plug-in tracks `mouseenter`, `mousepause`, `mousestart`, `mouseleave` and `click` events that happen within the bounds of a search result on a SERP [20]. Each mouse event is extended with the search query, a user ID and the ID of the corresponding result [20]. The resulting data packets are then sent to a specified key-value store at suitable intervals (Fig. 2) [20]. For integration, the developer has to specify jQuery selectors for (a) the HTML container element holding all results, (b) a single search result, (c) an element within a result holding the result ID and (d) links to landing pages.

The second function provided by the plug-in is intended for recording human relevance judgments, often also referred to as *conversions*, which are crucial for learning relevance models. It is realized as a JavaScript method that can be called from anywhere, e.g., upon clicking an upvote button next to a search result [20]. This method has to be provided with the value of the judgment (e.g., -1 for a downvote and $+1$ for an upvote) as well as the corresponding search query, session ID and user ID by the developer.

3.2 Preprocessor

After having been recorded using the above jQuery plug-in, all interaction data is received by SMR as a *stream of individual events* for preprocessing (Fig. 2). Additionally, information about a corresponding search session[5] is transferred when a user enters a SERP. These contain an anonymous user ID, the current search query and the ordered list of all results, among others [20]. Every event received by SMR is subsequently associated with its respective search session. This concept is referred to as a *collected search session.*

It is logically not possible to process events from search sessions that have not ended yet. Thus, all events are passed on in the SMR pipeline on a per–search session basis. Since it is unreliable to fire client-side `unload` events on

[5] For our purposes, a search session starts when entering and ends when leaving a SERP. For example, a reload triggers a new session, even for the same user and query.

a SERP, this is realized using a configurable time-out on the server side. For example, if no events related to a given session have been received for 2 minutes, it is considered finished and the collected search session is passed on for interaction feature computation (Fig. 2).

Moreover, the preprocessing component receives human relevance judgments that are required for learning actual models. These judgments are checked for validity, i.e., whether a corresponding search session exists during which the judgment happened. The latter is not the case if a judgment is triggered by a user who did not perform a search beforehand, e.g., because they received a link to a result from a friend. Relevance judgments are persisted at this point for later use by the Classification Processor (Fig. 2). Finally, for later filtering purposes, each valid judgment is associated with the list of queries triggered by the corresponding user ID.

3.3 Interaction Features Processor

The Interaction Features Processor is realized as a separate topology within our Storm-based system (Fig. 2). It receives *collected search sessions* from the preprocessor that are emitted as a stream by a dedicated spout. To ensure that all interaction events associated with a search session are ordered logically, invalid sequences of events are filtered out. This prevents the computation of faulty interaction feature values. An invalid sequence would be, e.g., if a `mouseleave` happens before a `mouseenter` event on the same search result. Typical causes for such a case can be faulty time stamps or latency while transferring data from client to server. Since at the moment we specifically focus on *mouse* interactions, search sessions that have been recorded on touch devices are eliminated as well.

Subsequently, the values of the actual interaction features are calculated per query–result pair. For example, the value of the *arrival time* is determined by subtracting the time stamp of the first `mouseenter` event on a result from the time stamp of the page load (which is available as meta information about the associated search session). The features we are considering are:

(i) ARRIVAL TIME, (ii) CLICKS (not leading to a landing page), (iii) CLICKTHROUGHS (leading to a landing page), (iv) CURSOR MOVEMENT TIME, (v) CURSOR SPEED (cursor trail divided by cursor movement time), (vi) CURSOR TRAIL, (vii) HOVERS, (viii) HOVER TIME, (ix) MAXIMUM HOVER TIME, (x) POSITION and (xi) UNCLICKED HOVERS (hovers during which no clickthrough happened).

These are in accordance with [20]. Features are moreover averaged over the number of hovers, if possible. This applies to *clicks, clickthroughs, cursor movement time, cursor trail, hover time* and *unclicked hovers* [20]. Finally, the computed values are persisted, which is important for later normalization purposes and actual use of SMR's relevance models (see below). In case feature values are already present for a query–result pair, they are automatically updated by adding the new values and taking the average over all values.

Within this topology, emitting a stream of collected search sessions is realized using a *spout*. Contrary, checking event sequence validity, the actual computation of feature values and updating values of already existing query–result pairs are realized through *bolts*.

The raw search sessions and associated events are not necessarily lost after they have been used for computing interaction features. Rather, SMR provides the option to persist all processed data. In this way, it is possible to batch-wise train a new model from parts of old data (e.g., after removing outdated information) before continuing to incrementally update this new model using real-time interactions and judgments.

3.4 Classification Processor

The Classification Processor is as well realized as a separate topology within our system (Fig. 2). It receives the previously calculated *interaction features* (one set per query–result pair) in terms of a stream that is emitted into the Storm cluster by a dedicated spout. Using the lists of queries associated to judgments during preprocessing, we filter out sets of interaction feature values that are not associated with a user who triggered at least one relevance judgment. This helps to ensure a good quality of our training data.

Moreover, relevance models provided by SMR highly depend on the layout of a SERP [20]. Thus, normalization of feature values is necessary to guarantee comparability between models related to different SERP layouts [20]. This happens in terms of dividing feature values by the maximum value of the respective feature across all results for the given query. Since interaction feature values arrive as a stream, maximum values change over time and have to be constantly updated. Hence, they become more precise the longer the system runs. This is a major difference compared to TMR, which—due to its batch-oriented nature—has look-ahead capabilities and knows exact maximum values from the start.

In the next step, we derive the normalized relevance rel_N for a query–result pair using the human relevance judgments that have been persisted in the preprocessing step. For this, all relevance judgments *judg* corresponding to the query–result pair (q,r) are summed up before dividing them by the sum of all judgments for the given query [20]:

$$\mathrm{rel_N}(q,r) = \frac{\sum\limits_{u \in U} \mathrm{judg}(u,q,r)}{\sum\limits_{s \in R} \sum\limits_{u \in U} \mathrm{judg}(u,q,s)} ,$$

with U the set of users who triggered a judgment and R the set of possible results for the query q. Normalizing judgments is important since otherwise, a result X that was among the results of 20 queries and received 10 positive judgments (rel_N=0.5) would be considered more relevant than a result Y that was among the results of only 5 queries and received 5 positive judgments (rel_N=1).

Having available interaction feature values and normalized relevance of a query–result pair, it is possible to use them as a training instance for SMR's relevance model. For this, the query–result pair is transformed into an instance that can be interpreted by the WEKA API [10]. The interaction features are labeled as attributes while "relevance" is labeled as the target attribute on which we train the model. At the moment, SMR has two built-in classifiers available that are provided by the WEKA API and trained in parallel. That is, a Hoeffding Tree, which is specifically aimed at incremental learning and is suitable for very large datasets [6], and an updateable version of Naïve Bayes[6], which also works for smaller datasets. The current states of the relevance models are serialized and persisted after each incremental update. These models are ready-to-use and can be instantly engaged for obtaining relevance predictions and feeding them back into a SERP for results optimization (Fig. 2). Moreover, all training instances are persisted to a file to enable manual inspections using, e.g., the WEKA GUI.

Within this topology, emitting a stream of interaction feature values is realized using a *spout*. Contrary, filtering and normalization tasks as well as incrementally training the relevance models are realized as *bolts*.

The incrementally trained relevance models are serialized and persisted after every update. This makes it possible to manually review the quality of the current model and interrupt or stop training if the model is reasonably stable, which helps to prevent overfitting. Moreover, SMR does not require to directly feed predictions by the incremental relevance model back into the ranking process of the underlying search engine. Rather, as just described, search engine owners are given the option to review the model before usage to ensure ranking quality.

3.5 Making Use of Relevance Models

SMR only caters for learning and providing relevance models. This means that the actual usage of a model is up to the search engine owner. A relevance model RM takes a vector of interaction feature values $\boldsymbol{I}$ for a given query–result pair (q,r) and returns a corresponding relevance prediction $\widehat{rel}$, i.e., $\mathrm{RM}(q, r, \boldsymbol{I}) = \widehat{rel}(q, r)$.

This can, e.g., be integrated into a scheduled process of updating search result ranking according to a learning-to-rank function that contains $\widehat{rel}$ as a parameter (Fig. 2). The interaction feature values used for prediction could be those recorded by SMR and persisted by the Interaction Features Processor.

4 Evaluation

To show SMR's capability of coping with realistic workloads, we have performed a large-scale log analysis of real-world user interactions. The anonymous data used were collected on two large hotel booking portals. We used the *number of conversions* (i.e., when a hotel has been actually booked by users) as relevance judgments for training our models. This stands in contrast to commonly used click models, where clicks are the prime indicators of relevance. First, we compare SMR to its analogous batch-wise approach TMR (cf. [20]) in terms of the

[6] `http://weka.sourceforge.net/doc.dev/weka/classifiers/bayes/NaiveBayesUpdateable.html` (2013-10-07).

prediction quality of the two systems. Second, we provide BBM (as a state-of-the art generative click model aiming at stream processing; cf. [16]) with the same set of raw interaction logs and compare its quality of relevance prediction against that of SMR. Third, we check SMR against a version of itself that considers clickthroughs only (SMR$_{\text{click}}$) as well as an analogous version of TMR, i.e., TMR$_{\text{click}}$. Results indicate that SMR is able to provide reasonably good relevance predictions that are not significantly different from those of TMR and might compare favorably to those of BBM—although the difference is not significant. Moreover, our system is superior to corresponding discriminative approaches that do not consider interactions other than clickthroughs. Subsequently, we have a look at the efficiency, robustness and scalability of the evaluated approaches. Results show that SMR can easily cope with realistic workloads in a manner that is robust to external influences. This is especially important in real-world settings with big data streams.

For detailed figures and descriptive statistics, see `http://vsr.informatik.tu-chemnitz.de/demo/SMR`. Also, we provide training data and serialized models for reproducing this evaluation using WEKA (cf. [10]).

4.1 Effectiveness

Method. Approximately 32 GB of raw tracking data were collected by SMR's interaction tracking facilities in May 2013 on two large hotel booking portals. Of these, ∼10 GB of interaction logs were chosen for evaluation, which correspond to ∼3.8 million search sessions over a period of 10 days. Based on these, we computed interaction features for a total of 86,915 query–result pairs. Because the collected data contained critical information about the cooperating company's business model, it was a requirement that all data was saved to a key-value store controlled by the company. In particular, we are not allowed to publish the concrete conversion–to–search session (CTS) ratio. Yet, it can be stated that this ratio is very low, i.e., *#conversions* $\ll$ *#search sessions*.

We divided the chosen raw interaction data into *10 distinct datasets DS0–DS9* (∼0.7–1.5 GB each) that were intended for training relevance models and corresponded to one day each. Since SMR cannot—due to its streaming-based nature—use fixed maximum values for interaction feature normalization (cf. Section 3.4), it produces different feature values for the same tracking data compared to TMR. Thus, processing the above raw datasets with both systems yields a total of 20 datasets containing interaction features and relevances (i.e., normalized conversions) of the extracted query–result pairs: DS^0_{TMR}–DS^9_{TMR} from TMR and DS^0_{SMR}–DS^9_{SMR} from SMR. For this, we considered only search sessions that were produced by users who triggered at least one conversion (in terms of booking a hotel). Conversions are treated as relevance judgments in analogy to [20], i.e., a greater number of conversions implies higher relevance and vice versa. For evaluating SMR, we *simulated a stream* of search sessions based on the logs containing raw interaction data.

In analogy to [20], we observed a very low ratio of booked hotels to search sessions. In addition with a high query diversity this leads to more than 99% of the query–result pairs having a relevance of either 0.0 or 1.0. Therefore, in this evaluation, we treat relevance prediction as a binary classification problem with two classes: “bad” (relevance < 0.5) and “good” (relevance ≥ 0.5). With more than 90% of the query–result pairs having a *bad* relevance and less than 10% having a *good* relevance, these classes are rather unbalanced. Thus, we use the *Matthews Correlation Coefficient* (MCC) for evaluations of model quality, which is suitable for cases with unbalanced classes [1].

Relevance models as provided by SMR and TMR are highly sensitive to layout specifics of the corresponding SERPs [20]. Yet, since the two hotel booking portals feature the exact same layout template, it is valid to use combined data from both portals for training the same model(s).

The Storm cluster used for evaluation was based on *Amazon EC2*[7]. It comprised four computing instances. An additional machine was used for logging purposes and hosting the database used. All computers in the Storm cluster were instances of type `m1.large`, featuring two CPUs and 7.5 GB RAM[8].

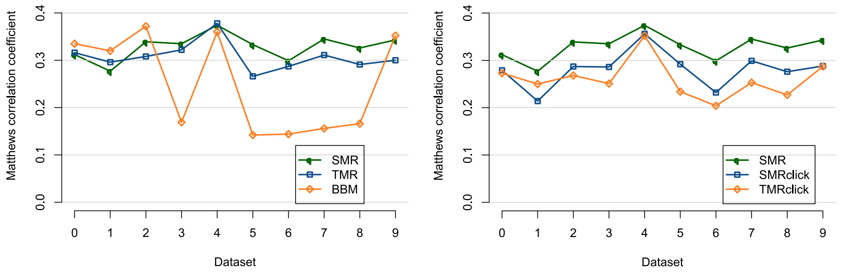


Fig. 3. MCC values for *DS0–DS9* (threshold = 0.5)

SMR vs. TMR. Based on the datasets described above, we trained a total of 20 Naïve Bayes classifiers (10 per system), as provided by TMR and SMR through the WEKA API. Thereby, our system used the updateable version of the classifier for incremental learning. The Naïve Bayes classifier was chosen because the amount of data available for evaluation was too small to train reasonably good Hoeffding Tree classifiers [6]. All classifiers learned have been evaluated using *10-fold cross validation*, from which we obtained corresponding MCC values. As can be seen in Fig. 3, the difference between SMR and TMR is not significant across the 10 datasets. This result has been validated using a Wilcoxon rank sum test, with $p{>}0.05$ ($\alpha{=}0.05$, $W{=}75$, 95.67% conf. int. = [-0.047, 0.004]). It implies that statistically, SMR yields the same prediction quality as TMR, even though it has less information available; particularly in terms of feature normalization and missing look-ahead capabilities. While Fig. 3 shows only MCC values at a

[7] `http://aws.amazon.com/ec2` (2013-09-30).

[8] `http://aws.amazon.com/en/ec2/instance-types/#instance-details` (2013-10-05).

threshold of 0.5, our result is underpinned by the exemplary receiver operating characteristic (ROC) curves depicted in Fig. 4, where SMR does not dominate TMR or vice versa. This is similar for the remaining nine datasets.

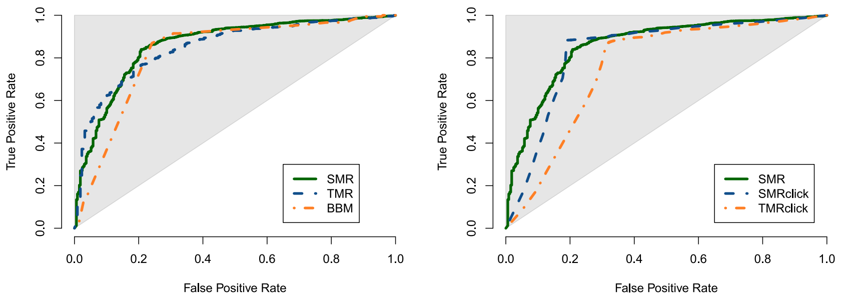


Fig. 4. ROC values for *DS7*

SMR vs. BBM. Additionally, we have compared SMR's prediction quality to that of a state-of-the-art generative click model designed for very large amounts of data and incremental learning. For this, we have used an existing re-implementation of BBM—as described in [16]—and provided it with the exact same raw interaction logs. Fig. 3 shows that BBM yields slightly better predictions for four out of ten datasets (*DS0*–*DS2*, *DS9*) at a threshold of 0.5 while SMR has a better prediction quality for the remaining six datasets. For this, predictions of BBM have been compared to the normalized relevances computed by SMR based on the available conversions. The difference between the two approaches is not significant according to a Wilcoxon rank sum test (α=0.05, W=64.5, p>0.05, 95.67% conf. int. = [-0.177, 0.021]). Still, our result indicates that SMR has the potential to provide relevance predictions that compare favorably to BBM. Particularly, Fig. 4 suggests that predictions of BBM can be partly dominated by SMR's predictions for certain datasets. We expect SMR's prediction quality to increase with amounts of data larger than used in this evaluation. Thus, we hypothesize that our system can predict relevance at least as good as BBM, whose predictions are being successfully used in industry.

SMR vs. SMR$_{\text{click}}$ vs. TMR$_{\text{click}}$. To investigate the influence of the additional user interactions, we have performed a comparison of SMR to versions of itself and TMR that consider clickthroughs only, named SMR$_{\text{click}}$ and TMR$_{\text{click}}$. Results show that SMR outperforms the click-only approaches across all 10 datasets (Fig. 3) based on *10-fold cross-validation*. Moreover, the MCC differences between SMR and SMR$_{\text{click}}$/TMR$_{\text{click}}$ are significant, as has been shown by two Wilcoxon rank sum tests (SMR$_{\text{click}}$: α=0.05, W=84.5, p<0.05, 95.67% conf. int. = [-0.075, -0.020]; TMR$_{\text{click}}$: α=0.05, W=90, p<0.01, 95.67% conf. int. = [-0.101, -0.044]). Our results are further supported by the ROC curves shown in Fig. 4, where SMR (area under ROC = 0.861) performs better than both SMR$_{\text{click}}$ (area under ROC = 0.834) and TMR$_{\text{click}}$ (area under ROC = 0.759).

These findings underpin that adding interaction data other than clicks yields considerable improvements for discriminative approaches, as has also been outlined in [11,13]. This is true even if clickthroughs show a correlation with relevance that is notably higher than those of the additional attributes (e.g., r=0.34 for DS^2_{TMR}).

4.2 Efficiency, Scalability and Robustness

Efficiency and Scalability. SMR is a feasible approach for processing web-scale interaction data. In contrast, TMR uses a batch-wise approach and non-incremental classifiers [20]. This means that all training data (in terms of query–result pairs, i.e., interaction features and relevances) already put into a model have to be re-processed for an update, which yields a time-complexity of $O(q) + O(s)$ with s = #search sessions in new log, q = #previously processed query–result pairs. Assume we receive one log with raw interaction data per day and want a daily model update. Then the amount of data that needs to be re-processed grows linearly. At some point, processing these data would take longer than 24 hours unless we add more/faster hardware to the system, which is, however, not a feasible approach in the long-term. Particularly, re-processing previously processed query–result pairs involves numerous slow database requests. To give just one concrete example from our evaluation, TMR needs ~5 hours for processing a single 1.5 GB log on a dual-core machine with a 2.3 GHz *Intel Core i5* CPU and 4 GB RAM. Since this corresponds to one day, processing the logs for two days would already take ~10 hours etc. This means that after *five days*, we exceed a processing time of 24 hours, which makes it impossible to provide a daily model update unless we use a better machine than the given one.

In contrast, SMR does not need to re-process logs from previous days since data is processed on a per–search session basis and models are learned incrementally. Thus, a model update considers only one search session at a time and the time-complexity of the update depends on the complexity of the classifier used. For example, "constant time per example [i.e., a query–result pair in our case]" [6] if using a Hoeffding Tree. SMR needs ~2 hours for processing all search sessions in a 1.5 GB log using the cluster described in Section 4.1. For this, the search sessions have been put into the system at the highest possible frequency. The log used corresponds to one day of real-world traffic from two hotel booking portals. This means that—using simple interpolation—SMR would be able to cope with approximately *12 times the load* based on the relatively simple cluster set-up used.

Finally, BBM has been specifically designed for incremental updates and web-scalability. As described in [16], 0.25 PB of data were processed using the generative click model. The authors state that it was possible to compute relevances for 1.15 billion query–result pairs in three hours on a MapReduce [5] cluster. BBM's time-complexity for updating a relevance model is $O(s)$.

Due to the differences in system architecture—TMR runs on a single node while the other two approaches require a cluster—the above is not an absolute, hardware-independent comparison of performance. Rather, it describes *relative* *performances* between the three systems. An overall, relative comparison of efficiency and scalability of the compared approaches is shown in Table 1.

Robustness. Being based on Storm, SMR is a highly robust system by design. In particular, it features guaranteed message passing[9] and high fault-tolerance[10] if one or more nodes die due to external reasons—which happened numerous times during our evaluation. In such a case, SMR continued processing the current interaction data from the step prior to the incident.

In [16], Liu et al. do not explicitly address the robustness of their approach. Rather, BBM has been designed for use as a MapReduce job on a *Hadoop* cluster. That is, differences in robustness between SMR and BBM originate from corresponding differences between Storm and Hadoop. Particularly, Hadoop has disadvantages when it comes to guaranteed message processing or when supervising/master nodes are killed.

Finally, TMR is the least robust of the compared approaches. In case the processing of a batch of data is stopped due to external reasons (e.g., a memory overflow), all data need to be re-processed. In particular, this means that already computed values of interaction features are useless since contributions of already processed data can not be subtracted out before starting over an iteration. Therefore, careful evaluation and set-up of the required hardware are necessary before using TMR to minimize the risk of costly and time-consuming errors.

4.3 Discussion and Summary

In this evaluation, we have shown that SMR does not perform significantly less effective than TMR, even though it relies on lower-quality information for training its relevance models. Moreover, SMR is more efficient, robust and scalable compared to its batch-wise predecessor. The difference of SMR's predictions to those of the generative state-of-the-art click model BBM were not significant as well. Yet, our results indicate that our discriminative approach can be advantageous over BBM for certain datasets and that it is more robust at similar efficiency and scalability. Finally, we have underpinned the value of interaction data other than clicks for relevance prediction, with clickthrough-only versions $\text{SMR}_{\text{click}}$ and $\text{TMR}_{\text{click}}$ performing significantly worse than SMR. However, there are some points remaining for discussion.

Discussion. *Why does SMR show the tendency to perform better than TMR, although its training data are of lower quality?* As described in Section 3.4, the maximum values for feature normalization change during the processing of a dataset due to SMR's streaming-based nature (i.e., no look-ahead is possible). This means that SMR has less information available and as a result, the training

[9] `https://github.com/nathanmarz/storm/wiki/Guaranteeing-message-processing` (2013-12-30).

[10] `https://github.com/nathanmarz/storm/wiki/Fault-tolerance` (2013-12-30).

data has lower quality. However, the different feature values for query–result pairs that appear early in a dataset can—purely by chance—lead to better predictions of SMR. This is especially the case because in this evaluation we were working with relatively small and closed datasets, as compared to a real-world setting. Hence, we strongly assume that in such a setting, the already non-significant difference between SMR and TMR would become even smaller.

Why does BBM make better predictions than SMR for DS2 but predicts worse for DS7? SMR computes almost the same amount of query–result pairs for the two datasets, with nearly identical means and distributions of the individual interaction features. In contrast, BBM has approximately 12% less search sessions available in *DS7* compared to *DS2*, which is due to the fact that search sessions are treated differently by BBM. Our system treats every page load event on a SERP as the beginning of a new search session. That is, if a user clicks a result and then returns to the SERP for clicking another result, SMR interprets this as two separate sessions. However, BBM handles this as a single search session with two clickthrough events. Besides containing more of these "combined" search sessions, *DS7* also features ∼12% less clickthrough events. All in all, this results in BBM having less data available for training its relevance model, which is an explanation for the lower-quality prediction compared to *DS2*. The same holds for other datasets showing similar differences, *DS2* and *DS7* are only used for representative purposes here.

Why are the MCC values relatively low (< 0.5*) in general?* The data collected for evaluation featured a very low CTS ratio, i.e., the amount of interaction data exceeded the available relevance judgments by far. To give just one example, the CTS ratios of both *DS0* and *DS1* lie under 1%, which is similar for the remaining datasets. This and the fact that the datasets used for evaluation were relatively small (compared to a realistic long-term scenario) leads to a rather low data quality. Yet, in an evaluation with larger amounts of data, we would expect increasing MCC values. This is, e.g., indicated in [20], where the authors work with datasets that are notably larger than 1.5 GB. Also, Huang et al. state that "adding more data can result in an order of magnitude of greater improvement in the system than making incremental improvements to the processing algorithms" [12].

How does SMR deal with click spam? Click spam is a major problem in systems where clicks are the main indicator for relevance [19]. However, in the specific setting we are focusing on in this paper, a high number of *conversions* indicates high relevance. Since conversions imply a confirmed payment, we do not have to deal with "traditional" click spam as described in [19]. Yet, in settings where no conversions are available, our discriminative approach has to rely on other indicators of relevance, such as clicks on social media buttons, for training its models. In such cases, additional measures have to be taken that prevent fraudulent behavior aiming at manipulating relevance models. Potential measures could be based on, e.g., filtering pre-defined behavior profiles, blacklists, personalized search [19] or the ranking framework described by [2].

Table 1. Overall relative comparison of the considered approaches

	effectiveness	efficiency	robustness	scalability
SMR	0	++	++	++
BBM	−	++	+	++
TMR (baseline)	*0*	*0*	*0*	*0*
$\text{SMR}_{\text{click}}$	−−	++	++	++
$\text{TMR}_{\text{click}}$	−−	0	0	0

Summary. Table 1 shows a comparison of all approaches considered in the evaluation. Since the systems—due to differences in the underlying architectures—are difficult to compare in an absolute, hardware-independent manner, we give a comparison of relative performances. Using TMR as the baseline, "0" indicates similar performance, "+"/"−" indicate a tendency and "++"/"−−" indicate a major or significant difference.

5 Limitations and Future Work

The following section discusses limitations of SMR and provides an overview of potential future work.

As described, in this paper SMR specifically aims at relevance prediction in the context of *travel search*. One specific feature of this setting is the fact that we can use hotel booking conversions as indicators of relevance. However, in a more general setting, other implicit or explicit relevance judgments are necessary. For example, one could obtain such judgments by providing optional vote up/down buttons to visitors or tracking clicks on Facebook "Like" buttons of a search result. Transferring SMR into such a more general context is our current work-in-progress.

Concerning the evaluation of our system, we had to rely on relatively small datasets compared to the real-world settings the system is intended for in the long-term. As part of our future work, we intend to evaluate SMR with larger datasets that simulate a real-world setting of a timespan considerably longer than 10 days. This will also give us the chance to investigate the performance of the Hoeffding Tree classifier, which becomes feasible only for very massive amounts of data [6].

Currently, SMR is only able to track client-side interactions on desktop PCs, i.e., mouse input. However, since the mobile market is steadily growing, an increasing number of users access search engines using their (small-screen) touch devices. This demands for also making use of touch interactions for predicting the relevance of results. Leveraging these valuable information is especially important for search engine owners and intended in future versions of SMR.

Finally, interaction features are often coupled with temporal features or their values change over time. This has to be addressed in the context of *concept drift* [21]. SMR is generally capable of handling changing data streams, as Tsymbal states that "[i]ncremental learning is more suited for the task of handling concept drift" [21]. However, the Naïve Bayes classifier used in the context of this paper would have to be replaced by an adequate concept drift–ready learner. A potential candidate is the CVFDT learner, which is based on Hoeffding trees and dismisses a subtree based on old data whenever a subtree based on recent data becomes more accurate [14].

6 Conclusions

This paper presented SMR, which is a novel approach to providing incremental models for predicting the relevance of web search results from real-time user interaction data. Our approach helps to ensure one of the *prime aspects of search engine quality*, i.e., providing users with the most relevant results for their queries. In contrast to numerous existing approaches, SMR does not require re-processing of already processed data for obtaining an up-to-date relevance model. Moreover, our system involves interaction features other than clicks and was specifically designed for coping with large amounts of data in real-time. This allows for feeding relevance predictions back into SERPs with relatively low latency.

For evaluating SMR, we have simulated a *real-world setting* with large amounts of interaction data from two *large hotel booking portals*. Comparison of our system to an analogous batch-wise approach showed that SMR is able to predict relevances that do not differ significantly, although it has less information available for training. Furthermore, we have compared the discriminative SMR approach to BBM—a generative state-of-the-art click model for incrementally processing big data streams that is successfully applied in industry. Results show that prediction quality does not differ significantly between the two systems. Still, they indicate that predictions by SMR might compare favorably to those of BBM, as it outperforms the click model for the majority of datasets. Additionally, we have considered a click-only version of SMR that was compared to the complete system. From the significantly better predictions of the latter, we conclude that interactions other than clicks yield valuable information for relevance prediction and should not be neglected.

As future work, we plan to adjust SMR to more general settings besides travel search. Moreover, it is planned to further optimize the system regarding performance and perform an evaluation with even larger amounts of real-world interaction data.

Acknowledgments. We thank Christiane Lemke and Liliya Avdiyenko for supporting us with their implementation of BBM. This work has been supported by the ESF and the Free State of Saxony.

References

1. Baldi, P., Brunak, S., Chauvin, Y., Andersen, C.A., Nielsen, H.: Assessing the accuracy of prediction algorithms for classification: an overview. Bioinformatics 16(5) (2000)

2. Bian, J., Liu, Y., Agichtein, E., Zha, H.: A Few Bad Votes Too Many? Towards Robust Ranking in Social Media. In: Proc. AIRWeb (2008)
3. Chapelle, O., Zhang, Y.: A Dynamic Bayesian Network Click Model for Web Search Ranking. In: Proc. WWW (2009)
4. Craswell, N., Zoeter, O., Tylor, M., Ramsey, B.: An Experimental Comparison of Click Position-Bias Models. In: Proc. WSDM (2008)
5. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. CACM 51(1) (2008)
6. Domingos, P., Hulten, G.: Mining High-Speed Data Streams. In: Proc. KDD (2000)
7. Dupret, G.E., Piwowarski, B.: A User Browsing Model to Predict Search Engine Click Data from Past Observations. In: Proc. SIGIR (2008)
8. Guo, F., Liu, C., Wang, Y.M.: Efficient Multiple-Click Models in Web Search. In: Proc. WSDM (2009)
9. Guo, Q., Agichtein, E.: Beyond Dwell Time: Estimating Document Relevance from Cursor Movements and other Post-click Searcher Behavior. In: Proc. WWW (2012)
10. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update. SIGKDD Explor. Newsl. 11(1) (2009)
11. Huang, J.: On the Value of Page-Level Interactions in Web Search. In: HCIR Workshop (2011)
12. Huang, J., White, R.W., Buscher, G., Wang, K.: Improving Searcher Models Using Mouse Cursor Activity. In: Proc. SIGIR (2012)
13. Huang, J., White, R.W., Dumais, S.: No Clicks, No Problem: Using Cursor Movements to Understand and Improve Search. In: Proc. CHI (2011)
14. Hulten, G., Spencer, L., Domingos, P.: Mining Time-Changing Data Streams. In: Proc. KDD (2001)
15. Joachims, T.: Optimizing Search Engines using Clickthrough Data. In: Proc. KDD (2002)
16. Liu, C., Guo, F., Faloutsos, C.: BBM: Bayesian Browsing Model from Petabyte-scale Data. In: Proc. KDD (2009)
17. Marz, N.: Storm Wiki, `http://github.com/nathanmarz/storm/wiki`
18. Navalpakkam, V., Churchill, E.F.: Mouse Tracking: Measuring and Predicting Users' Experience of Web-based Content. In: Proc. CHI (2012)
19. Radlinski, F.: Addressing Malicious Noise in Clickthrough Data. In: LR4IR Workshop at SIGIR (2007)
20. Speicher, M., Both, A., Gaedke, M.: TellMyRelevance! Predicting the Relevance of Web Search Results from Cursor Interactions. In: Proc. CIKM (2013)
21. Tsymbal, A.: The problem of concept drift: definitions and related work. Technical Report, Trinity College Dublin (2004)
22. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., Stoica, I.: Discretized streams: A fault-tolerant model for scalable stream processing. Technical Report, UC Berkeley (2012)

The Forgotten Many? A Survey of Modern Web Development Practices

Moira C. Norrie, Linda Di Geronimo, Alfonso Murolo, and Michael Nebeling

Department of Computer Science, ETH Zurich
CH-8092 Zurich, Switzerland
{norrie,lindad,amurolo,nebeling}@inf.ethz.ch

Abstract. With an estimated 21.9% of the top 10 million web sites running on WordPress, a significant proportion of the web development community consists of WordPress developers. We report on a survey that was carried out to gain a better understanding of the profile of these developers and their web development practices. The first two parts of the survey on the background and development practices were not exclusive to WordPress developers and therefore provide insight into general web developer profiles and methods, while the third part focussed on WordPress specifics such as theme development. We present the results of the survey along with a discussion of implications for web engineering research.

Keywords: web engineering, web development practices, WordPress developers.

1 Introduction

Second-generation content management systems (CMS) such as WordPress[1] and Drupal[2] are based on a crowdsourcing model where vast developer communities share themes and plugins. It is possible for endusers to create a web site without any programming effort by selecting an existing theme and adding content, even adding or customising the functionality through the user interface. At the same time, developers with programming skills and knowledge of the platform can create or edit PHP templates, CSS stylesheets and JavaScript functions to extend the functionality or create their own themes and/or plugins.

The availability of these platforms has radically changed the web development landscape with estimates that 21.9% of the top 10 million web sites are running on WordPress which has 60.3% of the CMS market share[3]. While many sites running on WordPress are personal web sites, the platform also supports everything from web sites created by professional designers for small businesses

[1] `http://www.wordpress.com`, `http://www.wordpress.org`
[2] `http://www.drupal.org`
[3] `http://w3techs.com/technologies/overview/content_management/all` (10.4.2014).

to large, complex sites created by teams of developers. WordPress has gone well beyond its origins as a blogging platform and its web sites include popular online newspapers, e.g. Metro UK[4], as well as e-commerce sites, e.g. LK Bennett[5].

Yet, WordPress and its developers have received little attention within the web engineering research community. Information gleaned from books about WordPress, online articles and forums as well as talking to personal contacts, suggests that many WordPress web sites are developed by individuals with a mix of technical and design skills. Books on developing WordPress themes such as [1] propose an *interface-driven* approach where the main steps are to develop a mockup of the interface, add client-side functionality and then migrate to the WordPress platform. This contrasts with the *model-driven* approaches [2] widely promoted within the web engineering research community.

Since WordPress developers form a significant part of the development community, we think it is important to get a better understanding of their development practices with a view towards identifying requirements and research challenges. We therefore decided to carry out a survey of web development practices which, although not exclusively limited to WordPress developers, made efforts to reach out to this community.

The results of our survey show that there is a need to support alternative methods to model-driven web engineering that are more in line with widely-used interface-driven practices and can be integrated with platforms such as WordPress. Further, since many developers seek inspiration from existing web sites and frequently reuse elements of design and implementation from other projects, a major issue is how to provide better support for reuse in all aspects of web engineering.

In Sect. 2, we discuss the background to this work including previous surveys of web development practices. Section 3 provides details of our survey and how it was carried out. The results are presented in three sections. Section 4 reports on results related to developer profiles in terms of experience, educational background and the size of team and organisation for which they work. Results on general methods and tools used in development are then presented in Sect. 5. The third part of the survey was specific to WordPress developers and we report on the results for this part in Sect. 6. Implications for web engineering research are discussed in Sect. 7, while concluding remarks are given in Sect. 8.

2 Background

The discipline of web engineering emerged in the late 1990s with calls for systematic methods for the development of web applications, e.g. [3,4]. This in turn led to the first of the ICWE series of conferences in 2001 and the appearance of the Journal of Web Engineering (JWE) in 2002. A position paper [5] in the first issue of JWE defined web engineering as "the application of systematic, disciplined and quantifiable approaches to the development, operation and maintenance of

[4] `http://metro.co.uk`

[5] `http://www.lkbennett.com`

web-based applications". The paper presented the characteristics of both simple and advanced web-based systems, discussing how the development of such systems differed from traditional software engineering. The authors concluded that "web engineering at this stage is a moving target since web technologies are constantly evolving, making new types of applications possible, which in turn may require innovations in how they are built, deployed and maintained."

It is certainly true that both web technologies and the kinds of web-based applications in everyday use have changed dramatically over the last decade. Further, the emergence of second-generation CMS such as WordPress which offer powerful platforms for both the development and operation of all kinds of web sites has also changed how a significant proportion of web sites are built, deployed and maintained. By offering a WordPress hosting platform[6], it is even possible for endusers to literally create and deploy a web site in a few clicks. Meanwhile, developers with technical skills and knowledge of the WordPress model can develop both plugins and themes offering rich functionality for their own use and to share with others.

Examining the research literature in web engineering over the past decade reveals less radical changes in proposals for how web sites should be developed. Model-driven approaches such as OOHDM [6], UWE [7], WebML [8] and WSDM [9] were introduced in the 1990s and early 2000s. Many of these still prevail although the modelling languages may have been extended to cater for new kinds of technologies and applications. For example, the web modelling language WebML has been extended to cater for service-enabled applications [10] and context-awareness [11]. The continued emphasis on model-driven approaches may be due to the fact that the main focus still appears to be on development within, or for, large enterprises using multi-disciplinary development teams involving programmers, database architects and graphic designers. In such settings, it might be expected that the model-driven approaches widely used in software engineering and information systems would be familiar to both programmers and database architects and hence adaptations for web engineering would be more likely to be adopted. However, it is interesting to note that in a recent paper analysing model-driven web engineering methodologies [2], they comment on the fact that model-driven web engineering approaches have still not been widely adopted and they accredit this mainly to the lack of tools.

There is little recent research literature reporting on modern web development practices, especially concerning the use of platforms such as WordPress. A number of surveys were carried out in the early 2000s in conjunction with the call for web engineering to be established as a discipline. Barry and Lang [12] reported on a study in Ireland on multimedia software development methods, which included web-based information systems. Almost a quarter (24.6%) reported that they did not use a methodology while the rest stated that they used an in-house variant, with most using what the researchers considered as outdated methods and only 6.2% using UML. Reasons given for not using methodologies were that

[6] `http://www.wordpress.com`

they were “too cumbersone”, “not suited to the real world” or “long training is required”.

Taylor et al. [13] carried out a study of web development activities in 25 UK organisations based on interviews. They found that few formalised techniques were used and most “web site development activities appeared to be undertaken in an ad hoc manner” with only 8 of the 25 using design techniques such as hierarchy charts, flowcharts and storyboards. They reported little or no use of established software development techniques. Around the same time, McDonald and Welland [14] carried out a study of web development practices based on in-depth interviews, in this case involving 9 UK organisations. Only 7 of the 15 interviewees claimed to have a development process in place, with only 2 of these 7 using industry standard software development processes. Although the majority of interviewees were using prototyping or user-centred design techniques, none of them mentioned involving endusers in validating the success of a project.

More recently, El Sheikh and Tarawneh [15] reported on a survey of web engineering practices in small Jordanian companies. The results of their study showed that many developers had 5 or fewer years of software experience and that the development processes were still mainly ad hoc, with little application of established web practices.

We wanted to find out how much the situation has changed over the years in terms of the profile of web developers and also the methods used. In particular, we were interested in the community of WordPress developers and whether their backgrounds, work settings and methods differ significantly from developers that use some form of web development framework rather than a CMS as the basis for their implementation.

3 Survey

The survey was designed to address both web developers and designers including those specifically developing with and for WordPress. We designed a questionnaire consisting of 31 questions distributed over three parts: *background*, *development practices* and *WordPress development*. We used a mix of 5-point Likert-scale questions for frequency-based answers or where agreement with different statements was to be expressed as well as open-ended questions.

The first part collected demographics by asking participants to provide their age, gender and country of residence and origin. We also enquired about any formal qualifications in computer science, design and web development. Other questions addressed the participant’s professional background and experience. We asked for the number of years working as a professional web developer, as well as the size of both the organisation they work for and their web development team. Participants were also encouraged to share any recent projects they developed and their role and specific contributions to the projects. These questions together enabled us to determine developer profiles that we will report in the next section.

The second part concerned their development practices with the goal of finding out about particular methods and tools used by participants. This part started with a question on how much they look at existing web sites for inspiration in the beginning of a project. Participants were asked how often they start by modifying an existing web site or theme as opposed to creating a new one from scratch. This was followed by questions on the use of sketching and digital mockups as well as the modelling of data and functional requirements. Participants were also asked to list any tools used for creating mockups and for modelling. These questions enabled us to better assess current development practices and identify trends between different groups of developers.

We also included a question on the reuse of resources published by other developers in terms of design or layout (HTML, CSS, etc.) and functionality (JavaScript, PHP, etc.). The goal was to get a better understanding of how different types of developers work and whether and how they make use of existing resources and material provided by other developers.

The second part closed with a question on the use of CMS such as WordPress or Drupal as opposed to web development frameworks as the starting point for web development. Participants were also asked to list the specific CMS and frameworks that they use. The answers to these questions were used to classify developers based on the software tools they typically use as the basis for development. These classifications were then used for comparison purposes in the analysis of other results.

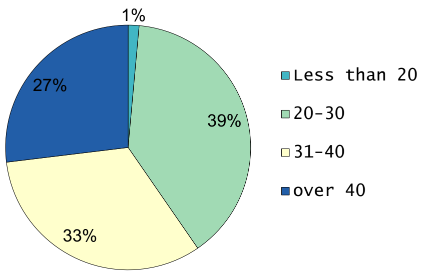
Finally, the third part specifically dealt with WordPress development. Only participants indicating that they were WordPress developers were asked to complete this part of the survey. We asked participants whether they mainly use an existing theme, modify an existing theme, create a child theme or create their own theme from scratch when they create a web site using WordPress. These questions tried to characterise the role of themes as one of the main concepts supporting reuse in WordPress.

The last set of questions allowed us to further profile WordPress developers and identify their specific needs and requirements. We asked how often they reuse code from previous WordPress projects and find themselves in the situation that they would like to mix parts of two or more themes. As before, we again distinguished between layout/style and functionality for mixing and matching parts. Finally, participants were asked to indicate the need for more customisation options of WordPress themes and which additional features they would like to see added in future versions of WordPress to support theme development.

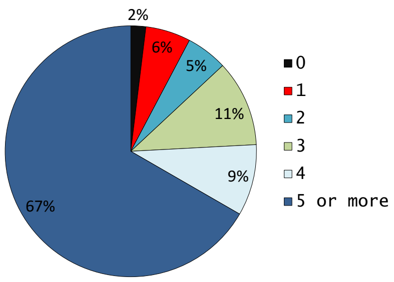
Before starting the online survey, we first asked members of our research group to fill it in and provide feedback on the design of the questionnaire. This allowed us to fix minor issues in the phrasing of some questions and calibrate the time typically required to answer all questions which was around 10 minutes. For dissemination, we primarily recruited via Twitter, reaching out to members of the web design and development community as well as the WordPress community, asking them to contribute to our survey and retweet our request for participation with a link to the online questionnaire. Targeted Twitter users ranged from users who frequently post and retweet links to articles related to web design and development to organisers of WordPress Meetup groups, giving us access to a network of several thousand followers of these active Twitter users. We also used Facebook and Reddit as well as directly contacting web developers known to us personally via email. Between January and February 2014, the survey was accessed 622 times and we received 208 complete responses that we included in the following analysis.

4 Developer Profiles

The 208 participants (83% male, 17% female) were from 24 different countries, with the majority living in the USA (49), Switzerland (45), Germany (39) or the UK (22). The age groups are shown in Fig. 1a and the years of professional web development in Fig. 1b.



(a) Age



(b) Years of experience

Fig. 1. Age and experience of participants

It is interesting to note that we had good coverage of the different age groups and the majority of our participants (67%) had 5 or more more years of experience as a professional developer. This contrasts with the survey of El Sheikh and Tarawneh [15] where 63% had 5 or fewer years of software experience.

Since one of the aims of our survey was to compare the profiles and methods of developers using a CMS as their main development platform with those using web development frameworks such as Django[7], Ruby on Rails[8] and Bootstrap[9], we asked participants how often they use each of these approaches.

The results in Fig. 2 show that the CMS developers are more likely to stick with this approach as 39% of participants answered that they always use this approach while only 18% always use a development framework. It is important

[7] `http://www.djangoproject.com`
[8] `http://rubyonrails.org`
[9] `http://getboostrap.com`

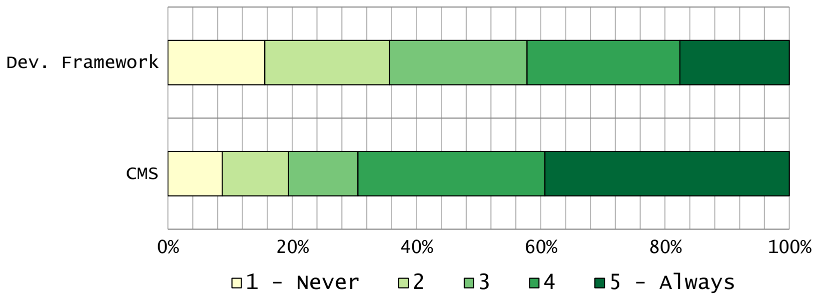


Fig. 2. Use of CMS or development framework

to note that these are not disjoint communities and 9% said that they always use both. 53% of the participants classified themselves as WordPress developers and 47% did not. Since we made efforts to target WordPress developers, it is not surprising that the majority were in this category, but we also achieved our aim to have a good mix of WordPress and non-WordPress developers. We note that some developers listed WordPress as a CMS that they use, but answered 'No' to the question asking if they are a WordPress developer. One reason for this might be that they interpreted the question as whether they are involved in developing the WordPress platform rather than whether they use it for developing applications. Another explanation could be that they classify themselves as endusers rather than developers since they create applications using the platform without actually doing any coding.

We classified the participants into three disjoint categories: those who answered 'Yes' to the question asking if they are a WordPress developer (WP), those who are not in WP but answered in the range 3-5 (sometimes to always) when asked if they use a CMS (CMS) and those who are not in WP and answered 1 or 2 (rarely or never) when asked if they use a CMS (Other). Thus developers who mostly use Drupal would be in the CMS category, while those who mainly use a web development framework and only occasionally use a CMS would be in the Other category. The sample sizes of each category are 111 (WP), 62 (CMS) and 35 (Other).

We asked participants how they would classify themselves in terms of whether they are designers, developers or both. Figure 3 reveals that, in all three categories, a significant proportion classified themselves as half-designer/half-developer (WP: 40%, CMS: 29%, Other: 34%), but there was also a significant proportion who classified themselves as 'developer' or 'mainly developer' (WP: 48%, CMS: 48%, Other: 63%). Since we were mainly targeting developer communities rather than design communities, we did not expect many participants to classify themselves as 'designer' or 'mainly designer'. Nevertheless, this shows that, rather than considering themselves as pure developers, many web development practitioners would see themselves as a mix of web developer and web designer.

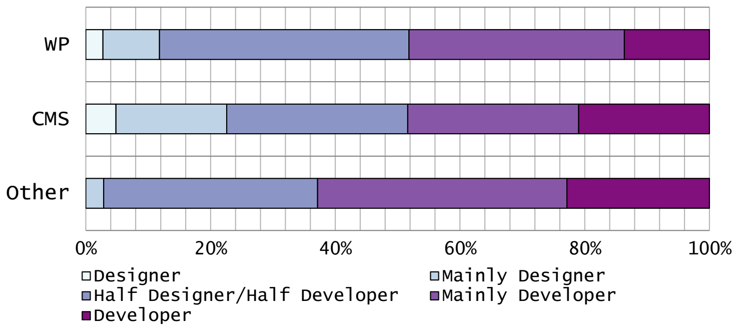


Fig. 3. Designer and/or developer

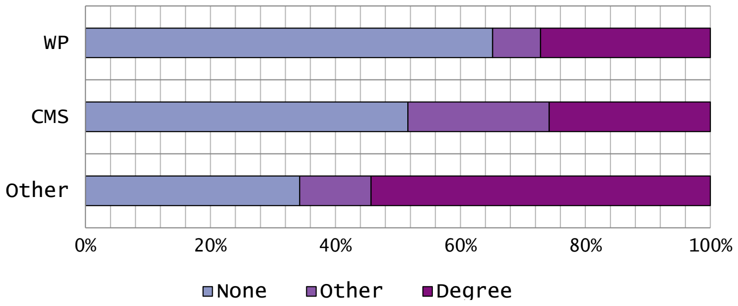


Fig. 4. Computer Science education

The educational background of the three categories is shown in Fig. 4 and 5. In all three categories, a significant proportion of participants have no formal qualification in computer science (WP: 65%, CMS: 52%, Other: 34%). Although we targeted developer communities rather than design communities, we also asked what, if any, qualification the participants have in design. As might be expected, relatively few have any formal qualification in design (WP: 31%, CMS: 25%, Other: 24%), although a significant proportion in each of the three categories classified themselves as half-designer/half-developer.

42% of all participants have no qualification in either computer science or design. In the case of participants who classified themselves as half-designer/half-developer, we also had 42% with no qualification in computer science or design. We also asked participants whether they have any kind of qualification in web development or specific web technologies. Taking this information into account, we still had 38% of participants with no formal education in computer science, design or web development.

Next, participants were asked to indicate the size of the organisation for which they work and also the size of their project team. The results shown in Fig. 6 indicate that a significant proportion of WordPress developers are either self-employed (42%) or belong to organisations with 5 or fewer employees (16%). On the other hand, it also shows that WordPress is not solely used by

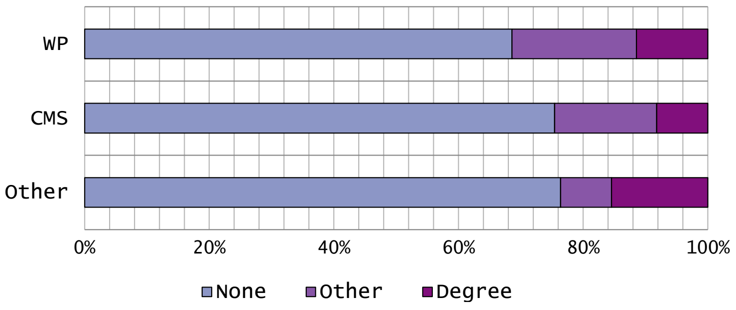


Fig. 5. Design education

individuals and small businesses since 10% of WordPress developers are working in organisations with more than 250 employees. While a significant proportion of non-CMS developers are also self-employed (14%) or in organisations with 5 or fewer employees (9%), the proportion working in organisations with more than 50 employees (51%) is far greater than for WordPress (20%).

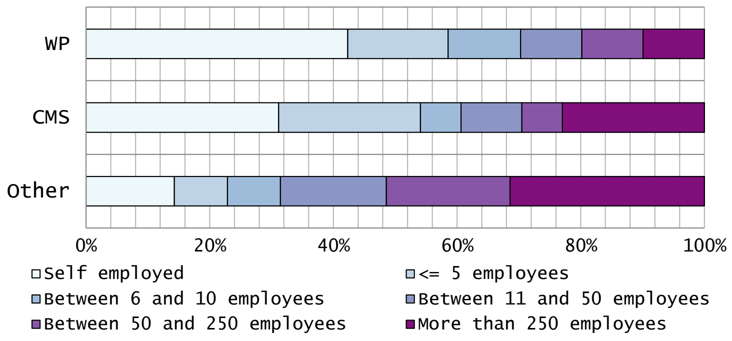


Fig. 6. Size of organisation

Previous surveys have tended to target organisations rather than individuals, and therefore have not involved developers who are self-employed. The smallest organisation involved in the survey by Taylor et al [13] had 20 employees. While the survey of El Sheikh and Tarawneh [15] targeted small companies in Jordan and 75% of companies had fewer than 10 employees, they also did not include self-employed developers.

Since many of our participants are self-employed or working in organisations with 5 or fewer members, clearly these developers either work alone or in very small teams and therefore the percentages for ‘no team’ and a team size of ‘5 or fewer’ would be expected to reflect this. Still, even in larger organisations, participants often work in small teams and 75% of WordPress developers work in teams with 5 or fewer members, with only 7% working in teams with more than 10 members. In the case of the non-CMS participants (Other), team sizes still tend to be small with 51% working in teams with 5 or fewer members, but 26% of them do work in teams with more than 10 members.

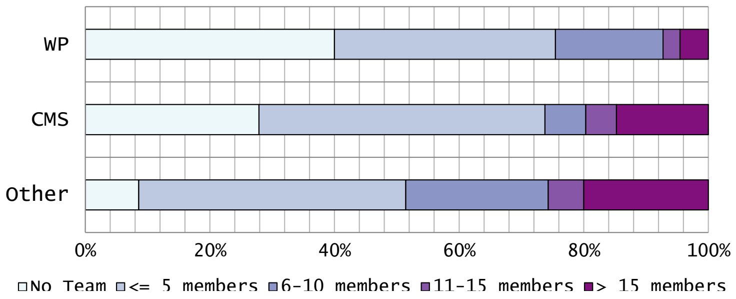


Fig. 7. Size of team

McDonald and Welland [14] estimated the average team size of web development projects in the organisations that they surveyed as 6. They argued that the small size of web development teams is one of the major differences to traditional software development teams, citing an article published in 2000 by Reifer [16] where he estimated the size of web development teams as 3-5 compared with traditional software development projects with hundreds of team members.

Interestingly, a survey of 200 Java developers carried out in 2011 by Munoz[10] reported that 40.7% worked in a team size of 1-5, 26.6% in a team size of 5-10 and 32.6% in teams larger than 10. These figures are actually not so different from the Other group where 52% work in teams of 1-5, 23% work in teams of 6-10 and 26% in teams larger than 10.

5 Methods and Tools

In this section, we report on the second part of the survey where all participants were asked questions about the methods and tools that they use in development projects. We started by asking them if they use existing web sites for inspiration at the beginning of a new project.

As shown in Fig. 8, more than 20% always look at examples of web sites for inspiration (WP: 23%, CMS: 24%, Other: 30%) and more than 50% answered 4 or 5 indicating that they often inspect examples (WP: 53%, CMS: 54%, Other: 67%). Although WordPress explicitly supports design-by-example through its notion of themes that can be easily accessed and previewed in online galleries, it is interesting to note that examples are used as much, if not more, in the Other group.

[10] `http://www.antelink.com/blog/software-developer-survey-first-chapter.html`

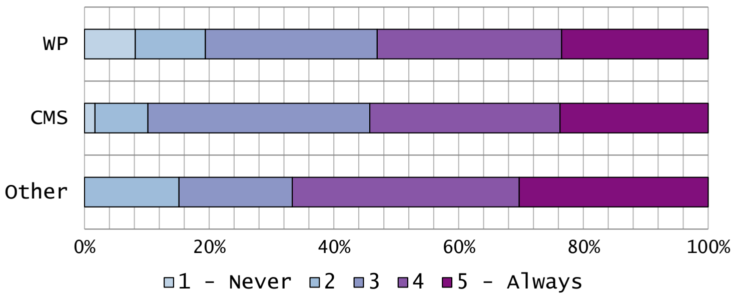


Fig. 8. Use of other web sites for inspiration

The survey by Taylor et al [13] published in 2002 also noted that examples of other web sites were often used for inspiration: "Roughly a third of those interviewed across 25 organisations studied indicated that they used other organisations' websites for design ideas in order to supplement their website design activities".

We also asked how often developers create a website or theme based on the modification of an existing web site or theme, either of their own or of another developer. As shown in Fig. 9, 61% of WordPress developers answered that they sometimes, often or always base the design of a web site or theme on an existing web site or theme. While none of the Other group answered that they always base a new design on an existing one, 50% said that they do this sometimes or often.

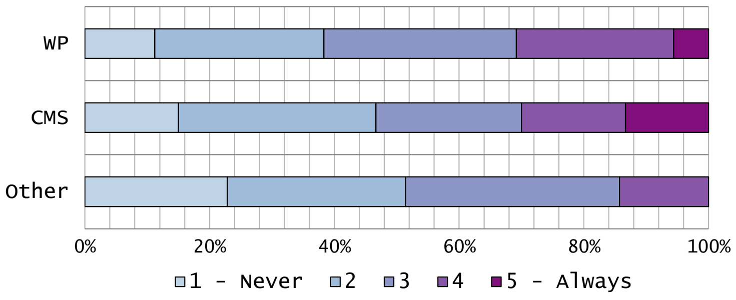


Fig. 9. Modify existing web sites or themes

Participants were asked how often they sketch mockups or create digital mockups and the results are shown in Fig. 10 and 11.

It is clear that sketching plays an important role with 43% of WordPress developers and 39% of the CMS group saying that they usually or always sketch. Sketching is used even more in the Other group with 57% stating that they usually or always sketch.

It is also common to produce digital mockups with more than 45% of the WordPress developers, 56% of the CMS group and 57% of the Other group answering 4 or 5 to indicate that they often or always use them. A range of tools

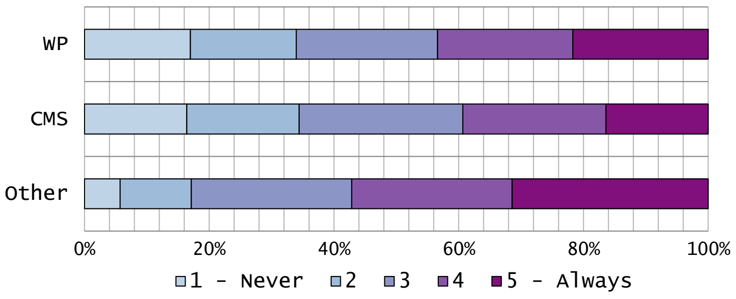


Fig. 10. Sketching mockups

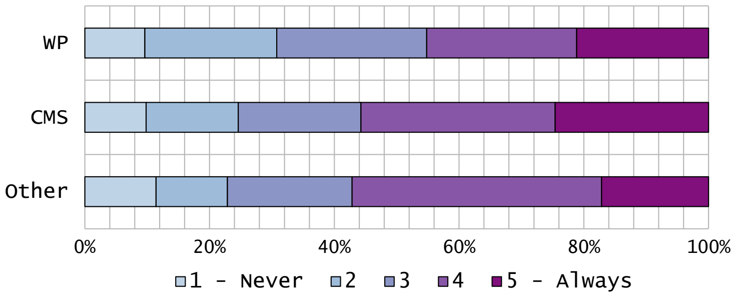


Fig. 11. Digital mockups

were listed including graphics editing tools such as Adobe Photoshop, diagram editors such as Microsoft Visio and wireframing tools such as Balsamiq[11].

Some developers wrote that they do not sketch or create digital mockups because they have a pure development role and implement the mockups produced by a graphic designer.

Since model-driven web engineering is widely promoted in the research community, we were interested in how frequently data and functional requirements are modelled. The results are shown in Fig. 12 and Fig. 13, respectively.

While only 28% of WordPress developers answered 4 or 5 to indicate that they often or always model data, 52% of Other developers answered that they often or always model data. The percentages answering that they often or always model functional requirements were also higher in the Other group (60%) compared to the WordPress developers (42%).

26% of participants answered that they never model data or functional requirements, leaving 74% who indicated that they use modelling at least some of the time. However, further analysis of the written comments provided by participants showed that the figures presented in Fig. 12 and 13 are very misleading as, in many cases, the participants had no idea what was meant by "modelling data" or "modelling functional requirements". We asked participants to list tools

[11] `http://balsamiq.com`

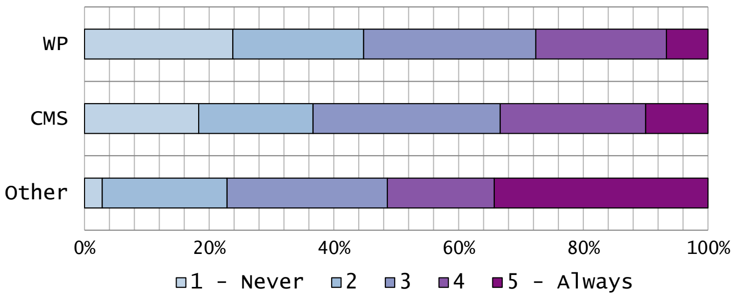


Fig. 12. Modelling data

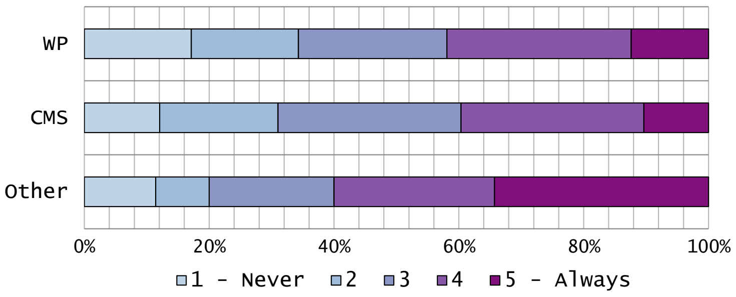


Fig. 13. Modelling functional requirements

that they use for modelling data and/or functional requirements and answers included "text documents", "spreadsheets and/or code editors", "Django to create prototypes" and "WordPress". One participant wrote "Not sure if I misunderstand this, but I usually just write requirements out—paper, text edit, google doc spreadsheets etc." Some listed project management tools and one participant even wrote something about testing and deployment. Only 11% of all participants listed an application or suite of tools that provides support for data or functional modelling. A further 5% wrote something general such as "paper and pen" or "whiteboard" that could also be considered as tools for modelling. This suggests that the number of developers actually doing some form of modelling of data or functional requirements is well below the figures reported.

This leads us to conclude that many of the participants are not even aware of software engineering practices, let alone applying them in even an informal way. This could be a consequence of the fact that a significant proportion of participants (WP 65%, CMS 52%, Other 34%) have no formal education in computer science.

6 WordPress Development Practices

The third part of the survey was only for WordPress developers as it deals specifically with the development of WordPress themes. A theme is a set of

PHP templates, CSS stylesheets and media objects that define the structure, navigation, functionality and presentation of a web site. The media objects included in a theme are generally static images used in the presentation of a web site such as the arrows used in sliders, buttons used in navigation and images that appear in the header. Endusers can select a theme from a gallery and create their own web site by simply adding content. A theme can also have a number of associated parameters to make it customisable through the general administrative interface. A professional developer will typically develop a theme to meet the requirements of a client, but they may also develop a theme for a particular class of clients such as restaurants, photographers or professional societies and make it customisable to the needs of a specific client.

The questions in this part of the survey were designed to find out more about how developers generate themes and specifically the forms of reuse that they employ or would like to have supported. Figure 14 presents an overview of the answers to a set of questions asking if and how they develop new themes for a specific project.

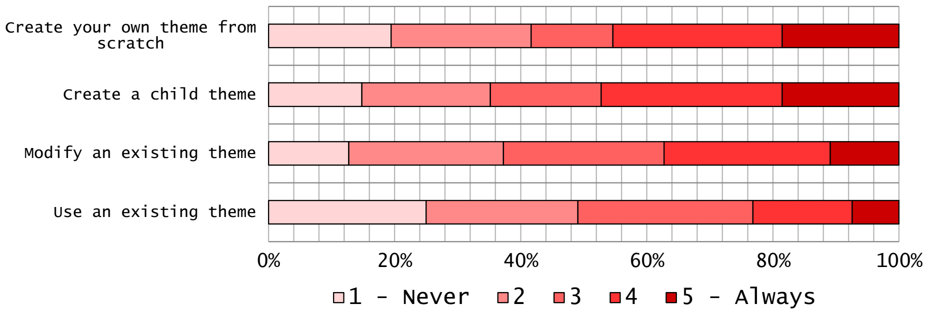


Fig. 14. Developing themes

7% of developers answered that they always use an existing theme. This means that these developers simply select a theme already provided by another developer and customise it for a client. This could involve the design of logos and other presentation features as well as the choice of layout, navigation and content.

19% of developers indicated that they always develop their own themes from scratch while 19% specified that they never do this. A developer can create a new theme based on an existing theme. This can be done by formally creating a child theme, but often developers will simply modify the PHP templates and CSS stylesheets provided. The results show that it is common for developers to build on existing themes using either of these approaches. 47% indicated that they often or always create a child theme of an existing theme for a project, while 37% answered that they often or always create a theme for a project by modifying an existing theme.

Since a theme defines an entire web site, a developer can only select a single theme as the starting point for a web site which means that they support all-or-nothing reuse. Once a theme has been selected, reuse of features from other themes can only be done by copying pieces of code and making any necessary modifications to integrate it into the theme under development.

We asked developers how often they find themselves in the situation where they would like to be able to mix parts of two or more existing themes. In the questions, we differentiated between the reuse of layout as specified by HTML and CSS and the reuse of functionality which could be either PHP code or JavaScript.

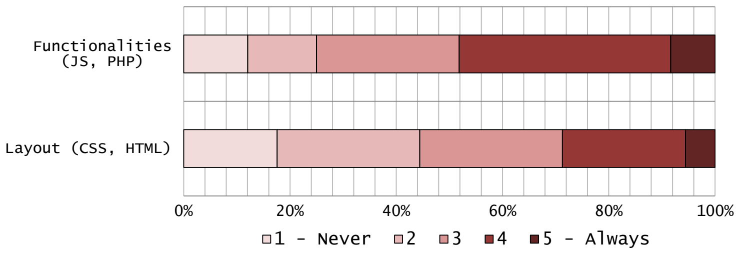


Fig. 15. Desire to be able to mix themes

Only 12% said they never find themselves in the situation where they would like to mix functionality from different themes, while 18% said they never want to mix layout. 75% answered 3-5 indicating that they sometimes, often or always find themselves wanting to mix functionality, while 56% answered 3-5 for layout.

We included a question asking participants to list what, if any, features they would like to see added to WordPress to support theme development. Most participants left this empty and the suggestions covered a range of issues from better means of managing media to easier ways of handling custom post types. One participant wrote: "I think where WordPress needs to go is to click-and-play development. Get rid of the need to code and it will take over the Internet". This comment can be interpreted as a request that it should go further in its support of enduser development.

7 Discussion

The results of our survey confirmed our impression that a significant proportion (40%) of WordPress developers work alone and act as both designer and developer. Since previous studies targeted organisations and tended to omit self-employed developers, it is impossible to say whether this is an increasing trend. However, the tendency for web developers to work in small teams as reported in earlier surveys is still the case, with 75% of WordPress developers working in teams with 1–5 members and only 26% of non-CMS developers working in teams with more than 10 members.

The fact that a significant proportion (40%) of both WordPress and non-CMS developers classified themselves as half-designer/half-developer, taken together with the fact that 41% of participants in this category have no qualification in either computer science or design, suggests that many of these developers have a mix of some design skills and some technical skills. Without a formal education in computer science and working alone or in very small organisations where there is likely to be a lack of in-house training, it may well be the case that many of these developers are not aware of modern software engineering methods, let alone using them. This would certainly be suggested by the answers that we received to our question about the tools that they use for the modelling of data and/or functional requirements. It is interesting to compare this with the results of the survey of Java developers carried out in 2011 by Munoz[12] where he reported that almost all participants had either a Bachelor or Masters degree in Computer Science.

This raises the question of whether efforts to adapt and promote software engineering methods, and specifically model-driven approaches, for web engineering are ever likely to have an impact in the web development community at large. Not only are many of these developers unaware of the underlying principles and techniques as well as the details of the methods, but many CMS developers have good reasons to employ interface-driven approaches rather than model-driven approaches. Therefore, while model-driven approaches may have their place in larger enterprises, we believe that the research community should also be exploring alternative methods that target practitioners at large.

One of the key findings of our study is how much developers build on the work of other designers and developers in their projects. This includes everything from using examples of other web sites for inspiration down to the detailed reuse of code. At the moment, there is little engineering support for reuse in CMS other than the concept of themes which support all-or-nothing reuse. Even the concept of child themes which is intended to provide a controlled way of developers building on existing themes is frequently not used and themes modified directly instead.

Within the web engineering research community, support for reuse has mainly been at the level of services. For example, WebComposition [17] allows applications to be built through hierarchical compositions of reusable application components. There has also been a lot of research in the area of web mashups to allow applications to be created through compositions of existing web sites, e.g. [18,19]. While this research is certainly relevant, the focus is purely on reuse rather on the design and development of new web sites as a whole and, as far as we know, there has been no attempt yet to adapt or integrate these methods into platforms such a WordPress. It is however important to mention that the work on mashups is also significant within the web engineering research community in its efforts to support enduser development.

Some researchers within the HCI community advocate a design-by-example approach [20,21] where the focus is very much on the reuse of the design aspects

[12] `http://www.antelink.com/blog/software-developer-survey-first-chapter.html`

of a web site. The idea is to allow users with little or no technical knowledge to develop their web site by selecting and combining elements of example web sites accessed in galleries. While the results of their studies are promising, they only deal with static elements and have not addressed the technical challenges of extracting and reusing functionality.

We believe that design-by-example is a promising paradigm worthy of detailed investigation within the web engineering research community. It is compatible with the interface-driven approaches that are currently in widespread use where mockups lead to prototypes that are gradually refined and migrated to platforms such as WordPress. With this goal in mind, we have started investigating how design-by-example could be supported in WordPress so that users could design and develop a fully functioning web site by selecting and reusing components of existing themes [22].

8 Conclusion

With a view to providing an insight into modern web practices, especially among the vast communities of WordPress developers, we have reported on the results of an online survey involving 208 participants working with CMS and/or web development frameworks. Unlike many previous surveys, we were keen to reach out to self-employed developers as well as developers within larger organisations and this we achieved.

The results point to the need for alternatives to model-driven approaches with a stronger focus on interface-driven development and enduser tools suited to the large numbers of developers with a lack of formal education in computer science and a mix of design and technical skills. Further, there is a need for methods that support the reuse of all aspects of web engineering and can be integrated into platforms such as WordPress that already have a significant proportion of the CMS market share and are continuing to grow.

Acknowledgements. We acknowledge the support of the Swiss National Science Foundation who financially supported this research under project FZFSP0_147257.

References

1. Silver, T.B., McCollin, R.: WordPress Theme Development - Beginner's Guide. Packt Publishing (2013)
2. Aragon, G., Escalona, M.-J., Lang, M., Hilera, J.R.: An Analysis of Model-Driven Web Engineering Methodologies. International Journal of Innovative Computing, Information and Control 9(1) (2013)
3. Coda, F., Ghezzi, G., Vigna, G., Garzotto, F.: Towards a Software Engineering Approach to Web Site Development. In: Proc. 9th Intl. Workshop on Software Specification and Design (1998)
4. Gellersen, H.W., Gaedke, M.: Object-Oriented Web Application Development. IEEE Internet Computing 3(1) (1999)

5. Deshpande, Y., Murugesan, S., Ginige, A., Hansen, S., Schwabe, D., Gaedke, M., White, B.: Web Engineering. Journal of Web Engineering 1(1) (2002)
6. Schwabe, D., Rossi, G.: The Object-Oriented Hypermedia Design Model. Communications of the ACM 38(8) (1995)
7. Koch, N., Kraus, A.: The Expessive Power of UML-based Web Engineering. In: Proc. 2nd Intl. Workshop on Web-Oriented Software Technology (IWWOST) (2002)
8. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): A Modeling Language For Designing Web Sites. Computer Networks 33(1-6), 137–157 (2000)
9. de Troyer, O., Leune, C.: WSDM: A User-Centred Design Method for Web Sites. Computer Networks and ISDN Systems 30(1-7) (1998)
10. Brambilla, M., Ceri, S., Fraternali, P., Acerbis, R., Bongio, A.: Model-Driven Design of Service-Enabled Web Applications. In: Proc. SIGMOD Industrial (2005)
11. Ceri, S., Daniel, F., Matera, M., Facca, F.M.: Model-Driven Development of Context-Aware Web Applications. TOIT 7(1) (2007)
12. Barry, C., Lang, M.: A Survey of Multimedia and Web Development Techniques and Methodology Usage. IEEE Multimedia (April–June 2001)
13. Taylor, M.J., McWilliam, J., Forsyth, H., Wade, S.: Methodologies and Web Site Development: A Survey of Practice. Information and Software Technology (44) (2002)
14. McDonald, A., Welland, R.: Web Engineering in Practice. In: Proc. 4th WWW Workshop on Web Engineering (2001)
15. Sheikh, A.E., Tarawneh, H.: A Survey of Web Engineering Practice in Small Jordanian Web Development Firms. In: Proc. 6th Joint Meeting on European Software Engineering Conference and ACM SIGSOFT Symposium on Foundations of Software Engineering (ESEC/FSE) (2007)
16. Reifer, D.: Web Development: Estimating Quick-to-Market Software. IEEE Software (November–December 2000)
17. Gellersen, H.-W., Wicke, R., Gaedke, M.: WebComposition: An Object-Oriented Support System for the Web Engineering Lifecycle. Computer Networks 29(8-13) (1997)
18. Daniel, F., Casati, F., Benatallah, B., Shan, M.C.: Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. In: Laender, A.H.F., Castano, S., Dayal, U., Casati, F., de Oliveira, J.P.M. (eds.) ER 2009. LNCS, vol. 5829, pp. 428–443. Springer, Heidelberg (2009)
19. Cappiello, C., Matera, M., Picozzi, M., Sprega, G., Barbagallo, D., Francalanci, C.: DashMash: A Mashup Environment for End User Development. In: Auer, S., Díaz, O., Papadopoulos, G.A. (eds.) ICWE 2011. LNCS, vol. 6757, pp. 152–166. Springer, Heidelberg (2011)
20. Hartmann, B., Wu, L., Collins, K., Klemmer, S.R.: Programming by a Sample: Rapidly Creating Web Applications with d.mix. In: Proc. 20th ACM User Interface Software and Technology Symposium (UIST) (2007)
21. Lee, B., Srivastava, S., Kumar, R., Brafman, R., Klemmer, S.: Designing with Interactive Example Galleries. In: Proc. Conf. on Human Factors in Computings Systems (CHI) (2010)
22. Norrie, M.C., Di Geronimo, L., Murolo, A., Nebeling, M.: X-Themes: Supporting Design-by-Example. In: Casteleyn, S., Rossi, G., Winckler, M. (eds.) ICWE 2014. LNCS, vol. 8541, Springer, Heidelberg (2014)

Using Path-Dependent Types to Build Type Safe JavaScript Foreign Function Interfaces

Julien Richard-Foy, Olivier Barais, and Jean-Marc Jézéquel

IRISA, Université de Rennes, France

Abstract. The popularity of statically typed programming languages compiling to JavaScript shows that there exists a fringe of the programmer population interested in leveraging the benefits of static typing to write Web applications. To be of any use, these languages need to statically expose the Web browser dynamically typed native API, which seems to be a contradiction in terms. Indeed, we observe that existing statically typed languages compiling to JavaScript expose the browser API in ways that either are not type safe, or when they are, typically over constrain the programmers. This article presents new ways to encode the challenging parts of the Web browser API in static type systems such that both type safety and expressive power are preserved. Our first encoding relies on type parameters and can be implemented in most mainstream languages but drags phantom types up to the usage sites. The second encoding does not suffer from this inconvenience but requires the support of dependent types in the language.

1 Introduction

We recently observed the emergence of several statically typed programming languages compiling to JavaScript (*e.g.* Java/GWT [10], Dart [9], TypeScript [8], Kotlin[1], Opa[2], SharpKit[3], Haxe [2], Scala [7], Idris [1], Elm [6]). Though dynamic typing has its merits and supporters, the mere existence of these statically typed languages shows that there is also a community interested in benefiting from static typing features (allowing *e.g.* better refactoring support in IDE, earlier error detection, etc.) to write Web applications. Nevertheless, at some point developers need a way to interface with the underlying Web browser dynamically typed native API using a *foreign function interface* mechanism.

We observe that, even though these languages are statically typed, their integration of the browser API either is not type safe or over constrain the programmers. Indeed, integrating an API designed for a dynamically typed language into a statically typed language can be challenging. For instance, the `createElement` function return type depends on the value of its parameter: `createElement('div')` returns a `DivElement`, `createElement('input')` returns an `InputElement`, *etc.*

[1] `http://kotlin.jetbrains.org`
[2] `http://opalang.org`
[3] `http://sharpkit.net`

Most of the aforementionned languages expose this function by making it return an `Element`, the least upper bound of the types of all the possible returned values, thus loosing type information and requiring users to explicitly downcast the returned value to its expected, more precise type. Another way to expose this function consists in exposing several functions, each one fixing the value of the initial parameter along with its return type: `createDivElement`, `createInputElement`, *etc.* are parameterless functions returning a `DivElement` and an `InputElement`, respectively. This encoding forces to hard-code the name of the to-be created element: it cannot anymore be a parameter. In summary, the first solution is not type safe and the second solution reduces the expressive power of the API.

This paper reviews some common functions of the browser API, identifies the patterns that are difficult to encode in static type systems and shows new ways to encode them in such a way that both type safety and expressive power are preserved. We show that type parameters are sufficient to achieve this goal and that *path-dependent types* provide an even more convenient encoding of the browser API.

The remainder of the paper is organized as follows. The next section reviews the most common functions of the browser API and how they are typically integrated into statically typed languages. Section 3 shows two ways to improve their integration such that type safety and expressiveness are preserved. Section 4 validates our contribution and discusses its limits. Section 5 discusses some related works and Section 6 concludes.

2 Background

This section reviews the most commonly used browser functions and presents the different integration strategies currently used by the statically typed programming languages GWT, Dart, TypeScript, Kotlin, Opa, SharpKit, Haxe, Scala, Idris and Elm.

All these languages support a foreign function interface mechanism, allowing developers to write JavaScript expressions from their programs. Since this mechanism is generally untyped and error prone, most languages (TypeScript, Kotlin, Opa, SharpKit, Haxe, Scala and Elm) support a way to define *external* typed interfaces. Most of them also expose the browser API this way, as described in the next section.

2.1 The Browser API and Its Integration in Statically Typed Languages

The client-side part of the code of a Web application essentially reacts to user events (*e.g.* mouse clicks), triggers actions and updates the document (DOM) according to their effect. Table 1 lists the main functions supported by Web

Table 1. Web browsers main functions that are challenging to encode in a static type system

Name	Description
`getElementsByTagName(name)`	Find elements by their tag name
`getElementById(id)`	Find an element by its `id` attribute
`createElement(name)`	Create an element
`target.addEventListener(name, listener)`	React to events

browsers according to the Mozilla Developer Network[4] (we omit the functions that can trivially be encoded in a static type system).

To illustrate the challenges raised by these functions, we present a simple JavaScript program using them and show how it can be implemented in statically typed programming languages according to the different strategies used to encode these functions. Listing 1 shows the initial JavaScript code of the program. It defines a function `slideshow` that creates a slide show from an array of image URLs. The function returns an image element displaying the first image of the slide show, and each time a user clicks on it with the mouse left button the next image is displayed.

```
function slideshow(sources) {
  var img = document.createElement('img');
  var current = 0;
  img.src = sources[current];
  img.addEventListener('click', function (event) {
    if (event.button == 0) {
      current = (current + 1) % (sources.length - 1);
      img.src = sources[current];
    }
  });
  return img
}
```

Listing 1. JavaScript function creating a slide show from an array of image URLs

The most common way to encode the DOM API in statically typed languages is to follow the standard interface specifications of HTML [18] and DOM [4].

The main challenge comes from the fact that the parameter types and return types of these functions are often too general. Indeed, functions `getElementsByTagName(name)`, `getElementById(id)` and `createElement(name)` can return values of type `DivElement` or `InputElement` or any other subtype of `Element` (their least upper bound). The interface of `Element` is more general and provides less features than its subtypes. For instance, the `ImageElement` type (representing images) has a `src` property that does not exist at the `Element` level. Similarly, the `MouseEvent` type has a `button` property that does not exist at the (more general) `Event` level, used by the function `addEventListener`.

[4] https://developer.mozilla.org/en-US/docs/DOM/DOM_Reference/Introduction

```
def slideshow(sources: Array[String]): ImageElement = {
  val img =
    document.createElement("img").asInstanceOf[ImageElement]
  var current = 0
  img.src = sources(current)
  img.addEventListener("click", event => {
    if (event.asInstanceOf[MouseEvent].button == 0) {
      current = (current + 1) % (sources.size - 1)
      img.src = sources(current)
    }
  })
  img
}
```

Listing 2. Scala implementation of `slideshow` using the standard HTML and DOM API

Listing 2 shows a Scala implementation of the `slideshow` program using an API following the standard specifications of HTML and DOM. The listing contains two type casts, needed to use the `src` property on the `img` value and the `button` property on the `event` value, respectively.

These type casts make the code more fragile and less convenient to read and write. That's why some statically typed languages attempt to provide an API preserving types as precisely as possible.

Of course, in the case of `getElementById(id)`, the `id` parameter does not give any clue on the possible type of the searched element, so it is hard to more precisely infer the return type of this function. Hence, most implementations use `Element` as their return type.

However, in the case of `getElementsByTagName(name)` and `createElement(name)`, there is exactly one possible return type for each value of the `name` parameter: *e.g.* `getElementsByTagName('input')` always returns a list of `InputElement` and `createElement('div')` always returns a `DivElement`. This feature makes it possible to encode these two functions by defining as many parameterless functions as there are possible tag names, where each function fixes the initial `name` parameter to be one of the possible values and exposes the corresponding specialized return type.

The case of `target.addEventListener(name, listener)` is a bit different. The `name` parameter defines the event to listen to while the `listener` parameter identifies the function to call back each time such an event occurs. Instead of being polymorphic in its return type, it is polymorphic in its `listener` parameter. Nevertheless, a similar property as above holds: there is exactly one possible type for the `listener` parameter for each value of the `name` parameter. For instance, a listener of `'click'` events is a function taking a `MouseEvent` parameter, a listener of `'keydown'` events is a function taking a `KeyboardEvent` parameter, and so on. The same pattern as above (defining a set of functions fixing the `name` parameter value) can be used to encode this function in statically typed languages.

```
def slideshow(sources: Array[String]): ImageElement = {
  val img = document.createImageElement()
  var current = 0
  img.src = sources(current)
  img.addClickEventListener { event =>
    if (event.button == 0) {
      current = (current + 1) % (sources.size - 1)
      img.src = sources(current)
    }
  }
  img
}
```

Listing 3. Scala implementation of `slideshow` using specialized functions

Listing 3 shows what would our `slideshow` implementation look like using such an encoding. There are two modifications compared to Listing 2: we use `document.createImageElement` instead of `document.createElement`, and we use `img.addClickEventListener` instead of `img.addEventListener`.

The `createImageElement` function takes no parameter and returns a value of type `ImageElement`, and the `addClickEventListener` function takes as parameter a function that takes a `MouseEvent` value as parameter, ruling out the need for type casts.

In the case of the `addEventListener` function we also encountered a slight variation of the encoding, consisting in defining one general function taking one parameter carrying both the information of the event name and the event listener.

```
img.addEventListener(ClickEventListener { event =>
  // ...
})
```

Listing 4. Implementation of `slideshow` using a general `addEventListener` function taking one parameter containing both the event name and the even listener

Listing 4 shows the relevant changes in our program if we use this encoding. The `addEventListener` function takes one parameter, a `ClickEventListener`, carrying both the name of the event and the event listener code.

Most of the studied languages expose the browser API following the standard specification, but some of them (GWT, Dart, HaXe and Elm) define a modified API getting rid of (or at least reducing) the need for downcasting, following the approaches described above.

2.2 Limitations of Existing Encoding Approaches

We distinguished three approaches to integrate the challenging parts of the browser API into statically typed languages. This section shows that each approach favours either type safety or expressive power but none provides both type safety *and* the same expressive power as the native browser API. We indeed consider that an API requiring users to do type casts is not type safe, while an API making it impossible to implement a function that can readily be implemented using the native API gives less expressive power to the programmer.

The first approach, consisting in using the least upper bound of all the possible types has the same expressive power as the native browser API, but is not type safe because it sometimes requires developers to explicitly downcast values to their expected specialized type.

The second approach, consisting in defining as many functions as there are possible return types of the encoded function, is type safe but leads to a less general API: each function fixes a parameter value of the encoded function, hence being less general. The limits of this approach are better illustrated when one tries to combine several functions. Consider for instance Listing 5 defining a JavaScript function `findAndListenTo` that both finds elements and registers an event listener when a given event occurs on them. Note that the event listener is passed both the event and the element: its type depends on both the tag name and the event name. This function cannot be implemented if the general functions `getElementsByTagName` and `addEventListener` are not available. The best that could be done would be to create one function for each combination of tag name and event name, leading to an explosion of the number of functions to implement. Thus, this approach gives less expressive power than the native browser API. Moreover, we find that defining many functions for the same task (creating a DOM element or listening to an event) clutters the API documentation: functions serving other purposes are hidden by these *same-purpose-functions*.

The third approach, consisting in combining two parameters into one parameter carrying all the required information, is type safe too, but reduces the expressive power because it forbids developers to partially apply the function by supplying only one parameter. Consider for instance Listing 6 that defines a function `observe` partially applying the `addEventListener` function[5]. Such a function cannot be implemented with this approach because the name of the event and the code of the listener cannot be decoupled. Thus, this one gives less expressive power than the native browser API.

In summary, the current integration of the browser API by statically typed languages compiling to JavaScript is either not type safe or not as expressive as the underlying JavaScript API. Indeed, we showed that our simple `slideshow` program requires type casts if the browser API is exposed according to the standard specification. We are able to get rid of type casts on this program by using modified browser APIs, but we presented two functions that we were not able to implement using these APIs, showing that they give less expressive power than the native API.

[5] The code of this function has been taken (and simplified) from the existing functional reactive programming libraries Rx.js [14] and Bacon.js (`http://baconjs.github.io/`).

```
function findAndListenTo(tagName, eventName, listener) {
  var elements = document.getElementsByTagName(tagName);
  elements.forEach(function (element) {
    element.addEventListener(eventName, function (event) {
      listener(event, element);
    });
  });
}
```

Listing 5. Combination of use of `getElementsByTagName` and `addEventListener`

```
function observe(target, name) {
  return function (listener) {
    target.addEventListener(name, listener);
  }
}
```

Listing 6. Partial application of `addEventListener` parameters

This article aims to answer the following questions: is it possible to expose the browser API in statically typed languages in a way that both reduces the need for type casts and preserves the same expressive power? What typing mechanisms do we need to achieve this? Would it be convenient to be used by end developers?

3 Contribution

In this section we show how we can encode the challenging main functions of the DOM API in a type safe way while keeping the same expressive power.

The listings in this paper use the Scala language, though our first solution could be implemented in any language with basic type parameters support, such as Java's *generics*[6]. Our second solution is an improvement over the first one, using *path-dependent types*.

3.1 Parametric Polymorphism

In all the cases where a type `T` involved in a function depends on the value of a parameter `p` of this function (all the aforementionned functions of the DOM API are in this case), we can encode this relationship in the type system using type parameters as follows:

1. Define a parameterized class `P[U]`
2. Set the type of `p` to `P[U]`
3. Use type `U` instead of type `T`
4. Define as many values of type `P[U]` as there are possible values for `p`, each one fixing its `U` type parameter to the corresponding more precise type

[6] For a lack of space, we do not present them here but all Java versions of all the Scala listings (excepted those using type members) are available online at `http://github.com/js-scala/js-scala/wiki/ICWE'14`

```
class ElementName[E]

trait Document {
  def createElement[E](name: ElementName[E]): E
  def getElementsByTagName[E](name: ElementName[E]): Array[E]
}

val Input = new ElementName[InputElement]
val Img = new ElementName[ImageElement]
// etc. for each possible element name
```

Listing 7. Encoding of the `createElement` function using type parameters

Listing 7 shows this approach applied to the `createElement` and `getElementsByTagName` functions which return type depends on their `name` parameter value: a type `ElementName[E]` has been created, the type of the `name` parameter has been set to `ElementName[E]` instead of `String`, and the return type of the function is `E` instead of `Element` (or `Array[E]` instead of `Array[Element]`, in the case of `getElementsByTagName`). The `ElementName[E]` type encodes the relationship between the name of an element and the type of this element[7]. For instance, we created a value `Input` of type `ElementName[InputElement]`.

Listing 8 shows the encoding of the `addEventListener` function. The `EventName[E]` type represents the name of an event which type is `E`. For instance, `Click` is a value of type `EventName[MouseEvent]`: when a user adds an event listener to the `Click` event, it fixes to `MouseEvent` the type parameter `E` of the `callback` function passed to `addEventListener`.

```
class EventName[E]

trait EventTarget {
  def addEventListener[E](
        name: EventName[E], callback: E => Unit): Unit
}

val Click = new EventName[MouseEvent]
val KeyUp = new EventName[KeyboardEvent]
// etc. for each possible event name
```

Listing 8. Encoding of the `addEventListener` function using type parameters

Listing 9 illustrates the usage of such an encoding by implementing our `slideshow` program presented in the introduction. Passing the `Img` value as a parameter to the `createElement` function fixes its `E` type parameter to `ImageElement`

[7] The type parameter `E` is also called a *phantom type* [12] because `ElementName` values never hold a `E` value.

```
def slideshow(sources: Array[String]) {
  val img = document.createElement(Img)
  var current = 0
  img.src = sources(current)
  img.addEventListener(Click, event => {
    if (event.button == 0) {
      current = (current + 1) % (sources.length - 1)
      img.src = sources(current)
    }
  })
  img
}
```

Listing 9. Scala implementation of the `slideshow` function using generics

so the returned value has the most possible precise type and the `src` property can be used on it. Similarly, passing the `Click` value to the `addEventListener` function fixes its `E` type parameter to `MouseEvent`, so the event listener has the most possible precise type and the `button` property can be used on the `event` parameter.

It is worth noting that this code is actually exactly the same as in Listing 2 excepted that type casts are not anymore required because the browser API is exposed in a way that preserves enough type information. Our way to encode the browser API is more type safe, but is it as expressive as the native API?

Listings 10 and 11 show how the challenging functions of Section 2.2, `findAndListenTo` and `observe`, can be implemented with our encoding. They are basically a direct translation from JavaScript syntax to Scala syntax, with additional type annotations.

```
def findAndListenTo[A, B](
      tagName: ElementName[A],
      eventName: EventName[B],
      listener: (A, B) => Unit) = {
  for (element <- document.getElementsByTagName(tagName)) {
    element.addEventListener(eventName, event => {
      listener(event, element)
    })
  }
}
```

Listing 10. Combination of `getElementsByTagName` and `addEventListener` functions encoded using type parameters

In summary, our encoding is type safe and gives as much expressive power as the native API since it is possible to implement exactly the same functions as we are able to implement in plain JavaScript.

```
def observe[A](target: EventTarget, name: EventName[A]) = {
  (listener: A => Unit) => {
    target.addEventListener(name, listener)
  }
}
```

Listing 11. Partial application of `addEventListener` encoded with type parameters

However, every function taking an element name or an event name as parameter has its type signature cluttered with phantom types (extra type parameters): the `observe` function takes a phantom type parameter `A` and the `findAndListenTo` function takes two phantom type parameters, `A` and `B`. These extra type parameters are redundant with their corresponding value parameters and they make type signatures harder to read and reason about.

3.2 Path-Dependent Types

This section shows how we can remove the extra type parameters needed in the previous section by using *path-dependent types* [16]. Essentially, the idea is to model type parameters using *type members*, as suggested in [17].

Programming languages generally support two means of abstraction: parameterization and abstract members. For instance Java supports parameterization for values (method parameters) and types (*generics*), and member abstraction for values (abstract methods). Scala also supports member abstraction for types through type members [5,16]. An abstract type member of a class is an inner abstract type that can be used to qualify values. Subclasses can implement and override their methods, and similarly they can define or refine their type mem-

```
trait ElementName {
  type Element
}

trait Document {
  def createElement(name: ElementName): name.Element
  def getElementsByTagName(
        name: ElementName): Array[name.Element]
}

object Div extends ElementName {
  type Element = DivElement
}
object Input extends ElementName {
  type Element = InputElement
}
// etc. for each possible element name
```

Listing 12. Encoding of `createElement` using path-dependent types

bers. A concrete subclass must provide a concrete implementation of its type members. Outside of the class, type members can be referred to using a type selection on an instance of the class: the type designator `p.C` refers to the `C` type member of the value `p` and expands to the `C` type member implementation of the singleton type of `p`.

```
trait EventName {
  type Event
}

object Click extends EventName { type Event = MouseEvent }

trait EventTarget {
  def addEventListener(name: EventName)
                      (callback: name.Event => Unit): Unit
}
```

Listing 13. Encoding of `addEventListener` using path-dependent types

Listings 12 and 13 show an encoding of `createElement`, `getElementsByTagName` and `addEventListener` in Scala using type members. Now, the `ElementName` type has no type parameter but a type member `Element`. The return type of the `createElement` function is `name.Element`: it refers to the `Element` type member of its `name` parameter. The `Div` and `Input` values illustrate how their corresponding element type is fixed: if one writes `createElement(Input)`, the return type is the `Element` type member of the `Input` value, namely `InputElement`. The same idea applies to `EventName` and `addEventListener`: the name of the event fixes the type of the callback.

The implementation of the `slideshow` function with this encoding is exactly the same as with the previous approach using generics. However, functions `findAndListenTo` and `observe` can be implemented more straightforwardly, as shown by listings 14 and 15, respectively.

With this encoding, the functions using event names or element names are not anymore cluttered with phantom types, and type safety is still preserved.

```
def findAndListenTo(eltName: ElementName, evtName: EventName)
     (listener: (evtName.Event, eltName.Element) => Unit) = {
  for (element <- document.getElementsByTagName(eltName) {
    element.addEventListener(evtName) { event =>
      listener(event, element)
    }
  }
}
```

Listing 14. Combination of `getElementsByTagName` and `addEventListener` using path-dependent types

```
def observe(target: EventTarget, name: EventName) =
  (listener: (name.Event => Unit)) => {
    target.addEventListener(name)(listener)
  }
```

Listing 15. Partial application of `addEventListener` using path-dependent types

4 Validation

4.1 Implementation in js-scala

We implemented our encoding in js-scala [11], a Scala library providing composable JavaScript code generators[8]. On top of that we implemented various samples, including non trivial ones like a realtime chat application and a poll application.

We have shown in this paper that our encoding leverages types as precisely as possible (our `slideshow` program is free of type casts) while being expressive enough to implement the challenging `findAndListenTo` and `observe` functions that were impossible to implement with other approaches.

4.2 API Clarity

We mentioned in the background section that a common drawback of existing approaches to bring more type safety was the multiplication of functions having the same purpose, making the API documentation harder to read.

Our encoding preserves a one to one mapping with browser API functions whereas existing approaches often have more than 30 functions for a same purpose. For instance, the `createElement` function is mapped by 31 specialized functions in GWT and 62 in Dart, the `addEventListener` is mapped by 32 specialized functions in GWT and 49 in Dart.

4.3 Convenience for End Developers

Statically typed languages are often criticized for the verbosity of the information they add compared to dynamically typed languages [15]. In our case, what is the price to pay to get accurate type ascriptions?

The first encoding, using type parameters, can be implemented in most programming languages because it only requires a basic support of type parameters (for instance Java, Dart, TypeScript, Kotlin, HaXe, Opa, Idris and Elm can implement it). However this encoding leads to cluttered type signatures and forces functions parameterized by event or element names to also take phantom type parameters.

However, the second encoding, using type members, leads to type signatures that are not more verbose than those of the standard specifications of the HTML

[8] Source code is available at `http://github.com/js-scala`

and DOM APIs, so we argue that there is no price to pay. However, this encoding can only be implemented in language supporting type members or dependent types (Scala and Idris).

4.4 Limitations

Our encodings only work with cases where a polymorphic type can be fixed by a value. In our examples, the only one that is not in this case is `getElementById`. Therefore we are not able to type this function more accurately (achieving this would require to support the DOM tree itself in the type system as in [13]).

Our solution is actually slightly less expressive than the JavaScript API: indeed, the value representing the name of an event or an element is not anymore a `String`, so it cannot anymore be the result of a `String` manipulation, like *e.g.* a concatenation. Fortunately, this case is uncommon.

5 Related Works

The idea of using dependent types to type JavaScript has already been explored by Ravi Chugh *et. al.* [3]. They showed how to make a subset of JavaScript statically typed using a dependent type system. However, their solution requires complex and verbose type annotations to be written by developers.

Sebastien Doreane proposed a way to integrate JavaScript APIs in Scala [7]. His approach allows developers to seamlessly use JavaScript APIs from statically typed Scala code. However, his work does not expose types as precise as ours (*e.g.* in their encoding the return type of `createElement` is always `Element`).

TypeScript supports overloading on constant values: the type of the expression `createElement("div")` is statically resolved to `DivElement` by the constant parameter value `"div"`. This solution is type safe, as expressive and as easy to learn as the native API because its functions have a one to one mapping. However, this kind of overloading has limited applicability because overload resolution requires parameters to be constant values: indeed, the `findAndListenTo` function would be weakly typed with this approach.

6 Conclusion

Having a statically typed programming language compiling to JavaScript is not enough to leverage static typing in Web applications. The native browser API has to be exposed in a statically typed way, but this is not an easy task.

We presented two ways to encode dynamically typed browser functions in mainstream statically typed languages like Java and Scala, using type parameters or path-dependent types. Our encodings give more type safety than existing solutions while keeping the same expressive power as the native API.

We feel that parametric polymorphism and, even more, dependent types are precious type system features for languages aiming to bring static typing to Web applications.

Acknowledgements. The authors would like to thank Thomas Degueule.

References

1. Brady, E.: Idris, a general-purpose dependently typed programming language: Design and implementation. Journal of Functional Programming 23(05), 552–593 (2013)
2. Cannasse, N.: Using haxe. The Essential Guide to Open Source Flash Development, 227–244 (2008)
3. Chugh, R., Herman, D., Jhala, R.: Dependent types for javascript. SIGPLAN Not. 47(10), 587–606 (2012)
4. W3C-World Wide Web Consortium et al.: Document object model (dom) level 3 core specification. W3C recommendation (2004)
5. Cremet, V., Garillot, F., Lenglet, S., Odersky, M.: A core calculus for scala type checking. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 1–23. Springer, Heidelberg (2006)
6. Czaplicki, E.: Elm: Concurrent frp for functional guis (2012)
7. Doeraene, S.: Scala.js: Type-Directed Interoperability with Dynamically Typed Languages. Technical report (2013)
8. Fenton, S.: Typescript for javascript programmers (2012)
9. Griffith, R.: The dart programming language for non-programmers-overview (2011)
10. Kereki, F.: Web 2.0 development with the Google web toolkit. Linux J., 2009(178) (February 2009)
11. Kossakowski, G., Amin, N., Rompf, T., Odersky, M.: JavaScript as an embedded DSL. In: Noble, J. (ed.) ECOOP 2012. LNCS, vol. 7313, pp. 409–434. Springer, Heidelberg (2012)
12. Leijen, D., Meijer, E.: Domain specific embedded compilers. ACM SIGPLAN Notices 35, 109–122 (1999)
13. Lerner, B.S., Elberty, L., Li, J., Krishnamurthi, S.: Combining Form and Function: Static Types for JQuery Programs. In: Castagna, G. (ed.) ECOOP 2013. LNCS, vol. 7920, pp. 79–103. Springer, Heidelberg (2013)
14. Liberty, J., Betts, P.: Reactive extensions for javascript. In: Programming Reactive Extensions and LINQ, pp. 111–124. Springer (2011)
15. Meijer, E., Drayton, P.: Static typing where possible, dynamic typing when needed: The end of the cold war between programming languages
16. Odersky, M., Cremet, V., Röckl, C., Zenger, M.: A nominal theory of objects with dependent types. In: Cardelli, L. (ed.) ECOOP 2003. LNCS, vol. 2743, pp. 201–224. Springer, Heidelberg (2003)
17. Odersky, M., Zenger, M.: Scalable component abstractions. ACM SIGPLAN Notices 40, 41–57 (2005)
18. Raggett, D., Le Hors, A., Jacobs, I., et al.: Html 4.01 specification. W3C Recommendation 24 (1999)

Visual vs. DOM-Based Web Locators: An Empirical Study

Maurizio Leotta[1], Diego Clerissi[1], Filippo Ricca[1], and Paolo Tonella[2]

[1] DIBRIS, Università di Genova, Italy
[2] Fondazione Bruno Kessler, Trento, Italy
{maurizio.leotta,filippo.ricca}@unige.it,
diego.clerissi@gmail.com, tonella@fbk.eu

Abstract. Automation in Web testing has been successfully supported by DOM-based tools that allow testers to program the interactions of their test cases with the Web application under test. More recently a new generation of visual tools has been proposed where a test case interacts with the Web application by recognising the images of the widgets that can be actioned upon and by asserting the expected visual appearance of the result.

In this paper, we first discuss the inherent robustness of the locators created by following the visual and DOM-based approaches and we then compare empirically a visual and a DOM-based tool, taking into account both the cost for initial test suite development from scratch and the cost for test suite maintenance during code evolution. Since visual tools are known to be computationally demanding, we also measure the test suite execution time.

Results indicate that DOM-based locators are generally more robust than visual ones and that DOM-based test cases can be developed from scratch and evolved at lower cost. Moreover, DOM-based test cases require a lower execution time. However, depending on the specific features of the Web application under test and its expected evolution, in some cases visual locators might be the best choice (e.g., when the visual appearance is more stable than the structure).

1 Introduction

The importance of test automation in Web engineering comes from the widespread use of Web applications (Web apps) and the associated demand for code quality. Test automation is considered crucial for delivering the quality levels expected by users [14], since it can save a lot of time in testing and it helps developers to release Web apps with fewer defects [1]. The main advantage of test automation comes from fast, unattended execution of a set of tests after some changes have been made to a Web app.

Several approaches can be employed to automate functional Web testing. They can be classified using two main criteria: the first concerns how test cases are developed, while, the second concerns how test cases localize the Web elements (i.e., GUI components) to interact with, that is what kind of *locators* (i.e., objects that select the target web elements) are used. Concerning the first criterion, it is possible to use the capture-replay or the programmable approach. Concerning the second criterion, there are three main approaches, Visual (where image recognition techniques are used to *locate* GUI components and a locator consists of an image), DOM-based (where Web page elements are *located* using the information contained in the Document Object Model and a locator is, for instance, an XPath expression) and Coordinates-based (where screen coordinates of the Web page elements are used to interact with the Web app under test). This categorization will be deeply analysed in the next section.

For developers and project managers it is not easy to select the most suitable automated functional web testing approach for their needs among the existing ones. For this reason, we are carrying out a long term research project aimed at empirically investigating the strengths and weaknesses of the various approaches (see also our previous work [9]).

In this work we evaluate and compare the visual and DOM-based approaches considering: the robustness of *locators*, the initial test suite development effort, the test suite evolution cost, and the test suite execution time. Our empirical assessment of the robustness of locators is quite general and tool independent, while the developers' effort for initial test suite development and the effort for test suite evolution were measured with reference to specific implementations of the two approaches. We instantiated such analysis for two specific tools, Sikuli API and Selenium WebDriver, both adopting the programmable approach but differing in the way they localize the Web elements to interact with during the execution of the test cases. Indeed, Sikuli API adopts the visual approach, thus using images representing portions of the Web pages, while Selenium WebDriver employs the DOM-based approach, thus relying on the HTML structure. We selected six open source Web apps and for each tool, we first developed a test suite per application and then we evolved them to a subsequent version. Moreover, since visual tools are known to be computational demanding, we also measured and compared the test suite execution time.

The paper is organized as follows: Sect. 2 gives some background on test case development using the visual and the programmable approaches, including examples for the two specific tools used in this work. In the same section, we describe test case repair activities. Sect. 3 describes our empirical study, reports the obtained results and discusses the pros and cons of the two considered approaches. We then present the related works (Sect. 4), followed by conclusions and future work (Sect. 5).

2 Background

There are several approaches for functional Web testing [13] and the choice among them depends on a number of factors, including the technology used by the Web app and the tools (if any) used for Web testing. Broadly speaking, there are two main criteria to classify the approaches to functional Web testing that are related to: (1) test case construction; and, (2) Web page element localisation.
For what concerns the first criterion, we can find two main approaches:

1) *Capture-Replay (C&R) Web Testing*: this approach consists of recording the actions performed by the tester on the Web app GUI and generating a script that provides such actions for automated, unattended re-execution.
2) *Programmable Web Testing*: this approach aims at unifying Web testing with traditional testing, where test cases are themselves software artefacts that developers write, with the help of specific testing frameworks. For Web apps, this means that the framework supports programming of the interaction with a Web page and its elements, so

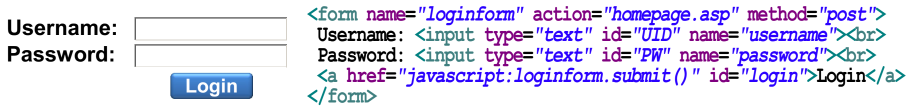
that test cases can, for instance, automatically fill-in and submit forms or click on hyperlinks.

An automated functional test case interacts with several Web page elements such as links, buttons, and input fields, and different methods can be employed to locate them. Thus, concerning the second criterion, we can find three different cases[1]:

1) *Coordinate-Based Localisation:* first generation tools just record the screen coordinates of the Web page elements and then use this information to locate the elements during test case replay. This approach is nowadays considered obsolete, because it produces test cases that are extremely fragile.
2) *DOM-Based Localisation:* second generation tools locate the Web page elements using the information contained in the Document Object Model. For example, the tools Selenium IDE and WebDriver employ this approach and offer several different ways to locate the elements composing a Web page (e.g., ID, XPath and LinkText).
3) *Visual Localisation:* third generation tools have emerged recently. They make use of image recognition techniques to identify and control GUI components. The tool Sikuli API belongs to this category.

In our previous work [9], we compared the capture-replay approach and the programmable approach using two 2nd generation tools: Selenium IDE and Selenium WebDriver. In this work, we fixed the test case definition method (i.e., programmable) and changed the locator type, with the aim of comparing the visual approach and the DOM-based approach. Empirical results refer to two specific programmable tools: Sikuli API and Selenium WebDriver.

Let us consider a running example, consisting of a typical login web page (login.asp). The login page requires the users to enter their credentials, i.e., *username* and *password* (see Fig. 1). After having inserted the credentials and clicked on "Login", the application shows the home page (homepage.asp). If credentials are correct, the username (contained in an HTML tag with the attribute `ID="uname"`) and the logout button are reported in the upper right corner of the home page (e.g., *John.Doe*). Otherwise, *Guest User* and login button are shown. For the sake of simplicity, the application does not report any error message in case of invalid credentials or unrecognised users.



```
<form name="loginform" action="homepage.asp" method="post">
 Username: <input type="text" id="UID" name="username"><br>
 Password: <input type="text" id="PW" name="password"><br>
 <a href="javascript:loginform.submit()" id="login">Login</a>
</form>
```

Fig. 1. login.asp – Page and Source

2.1 Programmable Web Testing

Programmable Web testing is based on manual creation of a test script. Web test scripts can be written using ad-hoc languages and frameworks or general purpose programming languages (such as Java and Ruby) with the aid of specific libraries able to play the role of the browser. Usually, these libraries extend the programming language with user friendly APIs, providing commands to, e.g., click a button, fill a field and submit

[1] `http://jautomate.com/2013/08/22/730/`

```
public class LoginPage {
 private final WebDriver driver;
 public LoginPage(WebDriver driver) {this.driver=driver;}
 public HomePage login(String UID, String PW) {
  driver.findElement(By.id("UID")).sendKeys(UID);
  driver.findElement(By.xpath("./input[2]")).sendKeys(PW);
  driver.findElement(By.linkText("Login")).click();
  return new HomePage(driver);
 }
}
```

```
public class HomePage {
 private final WebDriver driver;
 public HomePage(WebDriver driver)
   {this.driver = driver;}
 public String getUsername() {
  return
  driver.findElement(By.id("uname")).getText;
 }
}
```

Fig. 2. LoginPage and HomePage page objects in Selenium WebDriver

a form. Test scripts are completed with assertions (e.g., JUnit assertions if the language chosen is Java).

A best practice often used when developing programmable test cases is the *page object* pattern. This pattern is used to model the Web pages involved in the test process as objects, employing the same programming language used to write the test cases. In this way, the functionalities offered by a Web page become methods exposed by the corresponding page object, which can be easily called within any test case. Thus, all the details and mechanics of the Web page are encapsulated inside the page object. Adopting the *page object* pattern allows the test developer to work at a higher level of abstraction and it is used to reduce the coupling between Web pages and test cases, and the amount of duplicate code. For these reasons, the adoption of the *page object* pattern is expected to improve the test suite maintainability and evolvability [8].

DOM-Based Programmable Test Case Creation: The tool for Web app testing belonging to the DOM-based/Programmable category that we used in this work is Selenium WebDriver release 2.25.0 (in the following shortly referred to as WebDriver - http://seleniumhq.org/projects/webdriver/). WebDriver is a tool for automating Web app testing that provides a comprehensive programming interface used to control the browser. WebDriver test cases are implemented manually in a programming language (in our case Java) integrating WebDriver commands with JUnit or TestNG assertions. We chose WebDriver as the representative of this category, because: (1) it is a quite mature tool, (2) it is open-source, (3) it is one of the most widely-used open-source solutions for Web test automation (even in the industry), (4) during a previous industrial collaboration [8], we have gained a considerable experience on its usage.

As an example, we here use a simple WebDriver test case (Fig. 3 left): a successful authentication test case. It submits a valid login, using correct credentials (i.e., *username=John.Doe* and *password=123456*) and verifies that in the home page the user appears as correctly authenticated ("John.Doe" must be displayed in the top-right corner of the home page).

The first step is to create two page objects (LoginPage.java and HomePage.java) corresponding to the Web pages login.asp and homepage.asp respectively (see Fig. 2). The page object LoginPage.java offers a method to log into the application. This method takes username and password as inputs, inserts them in the corresponding input fields, clicks the Login button and returns a page object of kind HomePage.java, because after clicking the Login button the application moves to the page homepage.asp. HomePage.java contains a method that returns the username authenticated in the application or *"Guest"* when no user is authenticated. As shown in Fig. 2, the Web page elements

```
public void testLogin() { // WebDriver
 WebDriver driver = new FirefoxDriver();
 // we start from the 'login.asp' page
 driver.get("http://www.....com/login.asp");
 LoginPage LP = new LoginPage(driver);
 HomePage HP = LP.login("John.Doe","123456");
 // we are in the 'homepage.asp' page
 assertEquals("John.Doe", HP.getUsername());
}
```

```
public void testLogin(){ // Sikuli

 // we start from the 'login.asp' page
 CommonPage.open("http://www.....com/login.asp");
 LoginPage LP = new LoginPage();
 HomePage HP = LP.login("John.Doe", "123456");
 // we are in the 'homepage.asp' page
 assertTrue(HP.isUsernamePresent(new URL("JohnDoe.png")));
}
```

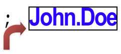

John.Doe

Fig. 3. TestLogin test case in Selenium WebDriver (left) and in Sikuli API (right)

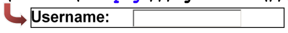
are located by searching for values in the DOM (using ID and LinkText locators) or navigating it (using XPath locators). While WebDriver offers several alternative ways to locate the Web elements in a Web page, the most effective one, according to WebDriver developers (http://docs.seleniumhq.org/docs/03_webdriver.jsp), is searching the elements by their ID values. Hence, whenever possible, we used this method. The second step is to develop the test case making use of the page objects (see Fig. 3 left). In the test method, first, a WebDriver of type FirefoxDriver is created, so that the test case can control a Firefox browser as a real user does; second, the WebDriver (i.e., the browser) opens the specified URL and creates a page object that instantiates LoginPage.java (modelling the login.asp page); third, using method login(...) offered by the page object, a new page object (HP) representing the page homepage.asp is created; finally, the test case assertion is checked.

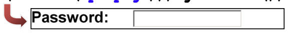
Visual Programmable Test Case Creation: The Web app testing tool, belonging to the Visual/Programmable category, that we used in this work is Sikuli API release 1.0.2 (in the following shortly referred to as Sikuli - http://code.google.com/p/sikuli-api/). Sikuli is a visual technology able to automate and test graphical user interfaces using screenshot images. It provides image-based GUI automation functionalities to Java programmers. We chose Sikuli as the representative of this category mainly because: (1) it is open-source and (2) it is similar to WebDriver, thus, we can create test cases and page objects similarly to the ones produced for WebDriver. In this way, using Sikuli, we are able to make the comparison between visual and DOM-based programmable tools fair and focused as much as possible on the differences of the two approaches. In fact, in this way we can use the same programming environment: programming language (Java), IDE (Eclipse), and testing framework (JUnit). Sikuli allows software testers to write scripts based on images that define the GUI widgets to be tested and the assertions to be checked. This is substantially different from the way in which WebDriver performs page element localisation and assertion checking.

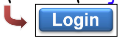
As an example, the Sikuli version of the `testLogin` test case is shown in Fig. 3 (right) while the related page objects are given in Fig. 4. The test case developed in Sikuli performs the same conceptual steps[2] as the WebDriver test case, as apparent from Fig. 3 (left) and Fig. 3 (right). The page objects (shown in Fig. 4) are instead quite different. To locate a Web page element, an instruction (based on the Sikuli Java API)

[2] Actually, in Sikuli there is no command to open Firefox at a specified URL as in WebDriver. We have encapsulated this functionality in a method, `CommonPage.open(...)`, that clicks the Firefox icon on the desktop, inserts the URL into the address bar and then clicks on the "go" arrow.

```
public class LoginPage {
 private ScreenRegion s = new DesktopScreenRegion();
 private Mouse m = new DesktopMouse();
 private Keyboard keyboard = new DesktopKeyboard();
 public HomePage login(String UID, String PW){
  m.click(s.find(new ImageTarget(new URL("un.png"))).getCenter());
  keyboard.type(UID);
  m.click(s.find(new ImageTarget(new URL("pw.png"))).getCenter());
  keyboard.type(PW);
  m.click(s.find(new ImageTarget(new URL("log.png"))).getCenter());
  return new HomePage();
 }
}
```







```
public class HomePage {
 private ScreenRegion s = new
               DesktopScreenRegion();
 private Mouse m = new DesktopMouse();
 public boolean isUsernamePresent(URL uname){
  try{m.click(s.find(new
     ImageTarget(uname)).getCenter());
   return true;
  } catch(Exception e) {return false;}
 }
}
```

Fig. 4. LoginPage and HomePage page objects in Sikuli API

is used which searches for the portion of Web page that looks like the image saved for the test suite (e.g., in a png file). Thus, in Sikuli, locators are always images. In addition, some other minor differences can be noticed in the test case implementation. For instance, in the case of WebDriver it is possible to assert that an element must contain a certain text (see the last line in Fig. 3 (left)), while in Sikuli it is necessary to assert that the page shows a portion equal to an image where the desired text is displayed (see the last line in Fig. 3 (right)).

2.2 Test Case Evolution

When a Web app evolves to accommodate requirement changes, bug fixes, or functionality extensions, test cases may become broken. For example, test cases may be unable to locate some links, input fields and submission buttons and software testers will have to repair them. This is a tedious and expensive task since it has to be performed manually (automatic evolution of test suites is far from being consolidated [11]).

Depending on the kind of maintenance task that has been performed on the target Web app, a software tester has to execute a series of test case repairment activities that can be categorised, for the sake of simplicity, in two types: *logical* and *structural*.

Logical Changes involve the modification of the Web app functionality. To repair the test cases, the tester has to modify the broken test cases and the corresponding page objects and in some cases, new page objects have to be created. An example of a change request that needs a logical repairment activity is enforcing the security by means of stronger authentication and thus adding a new Web page, containing an additional question, after the login.asp page of Fig. 1. In this case, the tester has to create a new page object for the Web page providing the additional authentication question. Moreover, she has to repair both the `testLogin` test cases shown in Fig. 3, adding a new Java command that calls the method offered by the new page object.

Structural Changes involve the modification of the Web page layout/structure only. For example, in the Web page of Fig. 1 the string of the login button may be changed to Submit. Usually, the impact of a structural change is smaller than a logical change. To repair the test cases, often, it is sufficient to modify one or more localisation lines, i.e., lines containing locators. In the example, the tester has to modify the LoginPage.java page object (see Fig. 2 and 4) by: (1) repairing the line:

```
driver.findElement(By.linkText("Login")).click()
```

in the case of Selenium WebDriver; or, (2) changing the image that represents the Login button in the case of Sikuli.

3 Empirical Study

This section reports the design, objects, research questions, metrics, procedure, results, discussion and threats to validity of the empirical study conducted to compare visual vs. DOM-based Web testing.

3.1 Study Design

The primary *goal* of this study is to investigate the difference in terms of robustness (if any) that can be achieved by adopting visual and DOM-based locators with the purpose of understanding the strengths and the weaknesses of the two approaches. Then, after having selected two tools that respectively belong to the two considered categories, as secondary *goal* we investigated the cost/benefit trade-off of visual vs. DOM-based test cases for Web apps. In this case, the *cost/benefit focus* regards the effort required for the creation of the initial test suites from scratch, as well as the effort required for their evolution across successive releases of the software. The results of this study are interpreted according to two *perspectives*: (1) *project managers*, interested in understanding which approach could lead to potentially more robust test cases, and in data about the costs and the returns of the investment associated with both the approaches; (2) *researchers*, interested in empirical data about the impact of different approaches on Web testing. The *context* of the study is defined as follows: two *human subjects* have been involved, a PhD student (the first author of this paper) and a junior developer (the second author, a master student with 1-year industrial experience as software tester); the *software objects* are six open source Web apps. The two human subjects participating in the study are referred below using the term "developers".

3.2 Web Applications

We have selected and downloaded six open-source Web apps from *SourceForge.net*. We have included only applications that: (1) are quite recent, so that they can work without problems on the latest versions of Apache, PHP and MySQL, technologies we are familiar with (actually, since Sikuli and WebDriver implement a black-box approach, the server side technologies do not affect the results of the study); (2) are well-known and used (some of them have been downloaded more than one hundred thousand times last year); (3) have at least two major releases (we have excluded minor releases because with small differences between versions the majority of the test cases are expected to work without problems); (4) belong to different application domains; and, (5) are non-RIA – Rich Internet Applications (to make the comparison fair, since RIAs can be handled better by the visual approach, see Sect. 3.7).

Table 1 reports some information about the selected applications. We can see that all of them are quite recent (ranging from 2008 to 2013). On the contrary, they are considerably different in terms of number of source files (ranging from 46 to 840) and number of lines of code (ranging from 4 kLOC to 285 kLOC, considering only the

Table 1. Objects: Web Applications from *SourceForge.net*

	Description	Web Site	1st Release				2nd Release			
			Vers.	Date	File[a]	kLOC[b]	Vers.	Date	File[a]	kLOC[b]
MantisBT	bug tracking system	sourceforge.net/projects/mantisbt/	1.1.8	2009	492	90	1.2.0	2010	733	115
PPMA[c]	password manager	sourceforge.net/projects/ppma/	0.2	2011	93	4	0.3.5.1	2013	108	5
Claroline	learning environment	sourceforge.net/projects/claroline/	1.10.7	2011	840	277	1.11.5	2013	835	285
Address Book	address/phone book	sourceforge.net/projects/php-addressbook/	4.0	2009	46	4	8.2.5	2012	239	30
MRBS	meeting rooms manager	sourceforge.net/projects/mrbs/	1.2.6.1	2008	63	9	1.4.9	2012	128	27
Collabtive	collaboration software	sourceforge.net/projects/collabtive/	0.65	2010	148	68	1.0	2013	151	73

[a] Only PHP source files were considered   [b] PHP LOC - Comment and Blank lines are not considered

lines of code contained in the PHP source files, comments and blank lines excluded). The difference in lines of code between 1st and 2nd release (columns 7 and 11) gives a rough idea of how different the two chosen releases are.

3.3 Research Questions and Metrics

Our empirical study aims at answering the following research questions, split between considerably tool-independent (A) and significantly tool-dependent (B) questions:

RQ A.1: *Do Visual and DOM-based test suites require the same number of locators?*
The goal is to quantify and compare the number of locators required when adopting the two different approaches. This would give developers and project managers a rough idea of the inherent effort required to build the test suites by following the two approaches. Moreover, the total number of locators could influence also the maintenance effort, since the more the locators are, the more the potential locators to repair could be. The metrics used to answer the research question is the number of created locators.

RQ A.2: *What is the robustness of visual vs. DOM-based locators?*
The goal is to quantify and compare the robustness of the visual and the DOM-based locators. This would give developers and project managers an idea of the inherent robustness of the locators created by following the two approaches. The metrics used to answer this research question is the number of broken locators.

RQ B.1: *What is the initial development effort for the creation of visual vs. DOM-based test suites?*
The goal is to quantify and compare the development cost of visual and DOM-based tests. This would give developers and project managers an idea of the initial investment (tester training excluded) to be made if visual test suites are adopted, as compared to DOM-based test suites. The metrics used to answer this research question is the time (measured in minutes) the two developers spent in developing visual test suites vs. DOM-based test suites.

RQ B.2: *What is the effort involved in the evolution of visual vs. DOM-based test suites when a new release of the software is produced?*
This research question involves a software evolution scenario. For the next major release of each Web app under test, the two developers evolved the test suites so as to make them applicable to the new software release. The test case evolution effort for visual and for DOM-based test suites was measured as the time (measured in minutes) spent to update the test suites, until they were all working on the new release.

RQ B.3: *What is the execution time required by visual vs. DOM-based test suites?* Image processing algorithms are known to be quite computation-intensive [2] and execution time is often reported as one of the weaknesses of visual testing. We want to quantitatively measure the execution time difference (in seconds) between visual and DOM-based test execution tools.

It should be noticed that the findings for research questions A.x are mostly influenced by the approaches adopted, independently of the tools that implement them, since the number of (DOM-based/visual) locators and the number of broken locators depend mostly on the test cases (and on the tested Web app), not on the tools. On the other hand, the metrics for research questions B.x (effort and execution time) are influenced by the specific tools adopted in the empirical evaluation.

3.4 Experimental Procedure

The experiment has been performed as follows:

– Six open-source Web apps have been selected from *SourceForge.net* as explained in Section 3.2.

– For each selected application, two equivalent test suites (written for Sikuli and WebDriver) have been built by the two developers, working in pair-programming and adopting a systematic approach consisting of three steps: (1) the main functionalities of the target Web app are identified from the available documentation; (2) each discovered functionality is covered with at least one test case (developers have assigned a meaningful name to each test case, so as to keep the mapping between test cases and functionalities); (3) each test case is implemented with Sikuli and WebDriver. For both approaches, we considered the following best practices: (1) we used the page object pattern and (2) we preferred, for WebDriver, the ID locators when possible (i.e., when HTML tags are provided with IDs), otherwise Name, LinkText, CSS and XPath locators were used following this order. Overall, for the WebDriver case we created: 82 ID, 99 Name, 65 LinkText, 64 CSS and 177 XPath locators. For each test suite, we measured the number of produced locators (to answer RQ A.1) and the development effort for the implementation as clock time (to answer RQ B.1). Each pair of test suites (i.e., visual and DOM-based) are equivalent because the included test cases test exactly the same functionalities, using the same sequences of actions (e.g., locating the same web page elements) and the same input data. The WebDriver test suites had been created by the same two developers about a year ago during a previous case study [9], while the Sikuli test suites have been created more recently, for the present study, which potentially gives a slight advantage to Sikuli (see *Threats to Validity* section).

– Each test suite has been executed against the second release of the Web app (see Table 1). First, we recorded the failed test cases (we highlight that no real bugs have been detected in the considered applications; all the failures are due to broken locators and minimally to modifications to the test cases logic) and then, in a second phase, we repaired them. We measured the number of broken locators (to answer RQ A.2) and the repair effort as clock time (to answer RQ B.2). Finally, to answer RQ B.3 we executed 10 times (to average over any random fluctuation of the execution time) each of the 12 repaired test suites (both WebDriver and Sikuli test cases) and recorded the execution times. We executed the test suites on a machine hosting an Intel Core i5 dual-core processor (2.5 GHz) and 8 GB RAM, with no other computation or communication load, in order to avoid CPU or memory saturation. To avoid as much as possible network delays we installed the web server hosting the applications on a machine belonging to the same LAN.

– The results obtained for each test suite have been compared to answer our research questions. On the results, we conducted both a quantitative analysis and a qualitative analysis, completed with a final discussion where we report our lessons learnt. The test suites source code can be found at: http://softeng.disi.unige.it/2014-Visual-DOM.php

3.5 Quantitative Results

This section reports the quantitative results of the empirical study, while the reasons and implications of the results are further analysed in Section 3.6.

RQ A.1. Table 2 shows the data to answer the A.x research questions by reporting, for each application, the number of visual and DOM-based broken/total locators. The number of locators required to build the test suites varies from 81 to 158 when adopting the visual approach and from 42 to 126 when adopting the DOM-based one. For all the six applications the number of required locators is higher when adopting the visual approach. Considering the data in aggregate form, we created 45% more Visual locators than DOM locators (706 visual vs. 487 DOM-based locators). To summarise, with respect to the research question RQ A.1, the visual approach has always required to create a higher number of locators.

RQ A.2. From the data shown in Table 2, we can see that in only two test suites out of six visual locators result more robust than the DOM-based ones (i.e., Mantis and Collabtive), while in the remaining four cases the DOM-based locators are more robust. Overall, 312 visual locators out of 706 result broken, while only 162 DOM-based locators out of 487 have been repaired (i.e., 93% more broken locators in the case of the visual approach). To summarise, with respect to RQ A.2, the result is not clear-cut. Generally, DOM-based locators are more robust but in certain cases (i.e., depending on the kind of modifications among the considered releases), visual locators proved to be the most robust (e.g., in Collabtive only 4 broken visual locators vs. 36 DOM-based ones).

RQ B.1. Table 3 reports some data about the developed test suites. For each application, it reports: the number of test cases and page objects in the test suites built for the newer release of each application (c. 1- 2); the time required for the initial development of the test suites (c. 3- 4); the percentage difference between the initial development time required by WebDriver vs. Sikuli (c. 5); the p-value of the Wilcoxon paired test used

Table 2. Visual vs. DOM-Based Locators Robustness

	Visual		DOM-Based	
	Broken Locators	Total Locators	Broken Locators	Total Locators
MantisBT	15	127	29	106
PPMA	78	81	24	42
Claroline	56	158	30	126
Address Book	103	122	14	54
MRBS	56	83	29	51
Collabtive	4	135	36	108

Table 3. Test Suites Development

	Number of		Time (Minutes)				Code (Java LOC)					
	Test Cases	Page Objects	Sikuli API	Web Driver		p-value	Sikuli API			WebDriver		
							Test	PO	Total	Test	PO	Total
MantisBT	41	30	498	383	-23%	**< 0.01**	1645	1291	**2936**	1577	1054	**2631**
PPMA	23	6	229	98	-57%	**< 0.01**	958	589	**1547**	867	346	**1213**
Claroline	40	22	381	239	-37%	**< 0.01**	1613	1267	**2880**	1564	1043	**2607**
Address Book	28	7	283	153	-46%	**< 0.01**	1080	686	**1766**	1078	394	**1472**
MRBS	24	8	266	133	-50%	**< 0.01**	1051	601	**1652**	949	372	**1321**
Collabtive	40	8	493	383	-22%	**< 0.01**	1585	961	**2546**	1565	650	**2215**

to assess whether the development time difference is statistically significant (c. 6) and finally the test suites size (measured in Lines Of Code (LOC), comment and blank lines have not been not considered), split between page objects and test cases, for the newer release of each application (c. 7-12). The development of the Sikuli test suites required from 229 to 498 minutes, while the WebDriver suites required from 98 to 383 minutes. In all the six cases, the development of the WebDriver test suites required less time than the Sikuli test suites (from 22% to 57%). This is related with the lower number of locators required when adopting the DOM-based approach (see RQ A.1). According to the Wilcoxon paired test (see c. 6 of Table 3), the difference in test suite development time between Sikuli and WebDriver is statistically significant (at $\alpha = 0.05$) for all test suites. For what concerns the size of the test suites (Table 3, c. 9, *Total*), we can notice that in all the cases, the majority of the code is devoted to the test case logics, while only a small part is devoted to the implementation of the page objects. Moreover, the number of LOCs composing the test cases is very similar for both Sikuli and WebDriver, while it is always smaller in WebDriver for what concerns the page objects (often, in the Sikuli page objects, two lines are needed to locate and perform an action on a web element while in WebDriver just one is sufficient, see for example Fig. 2 and 4). To summarise, with respect to the research question RQ B.1 we can say that for all the six considered applications, the effort involved in the development of the Sikuli test suites is higher than the one required by WebDriver.

RQ B.2. Table 4 shows data about the test suites repairing process. In detail, the table reports, for each application, the time required to repair the test suites (Sikuli and WebDriver), and the number of repaired test cases over the total number of test cases. The WebDriver repair time is compared to the Sikuli repair time by computing the percentage difference between the two and by running the Wilcoxon paired test, to check for statistical significance of the difference. Sikuli test suites required from 7 to 126 minutes to be repaired, while WebDriver test suites required from 46 to 95 minutes. The results are associated with the robustness of the two kinds of locators (RQ A.2) employed by the two tools and thus follow the same trend: in four cases out of six, repairing of the

Table 4. Test Suites Maintenance

	Sikuli API		WebDriver			
	Time	Test	Time			Test
	Minutes	Repaired	Minutes		p-value	Repaired
MantisBT	76	37 / 41	95	+ 25%	**0.04**	32 / 41
PPMA	112	20 / 23	55	- 51%	**< 0.01**	17 / 23
Claroline	71	21 / 40	46	- 35%	0.30	20 / 40
Address Book	126	28 / 28	54	- 57%	**< 0.01**	28 / 28
MRBS	108	21 / 24	72	- 33%	**0.02**	23 / 24
Collabtive	7	4 / 40	79	+ 1029%	**< 0.01**	23 / 40

Table 5. Test Suites Execution

	Number of Test Cases	Sikuli API			WebDriver				
		Mean	σ		Mean			σ	
		Seconds	Absolute	Relative	Seconds		p-value	Absolute	Relative
MantisBT	41	2774	60	2,2%	1567	- 43%	**< 0.01**	70	4,5%
PPMA	23	1654	12	0,7%	924	- 44%	**< 0.01**	35	3,8%
Claroline	40	2414	34	1,4%	1679	- 30%	**< 0.01**	99	5,9%
Address Book	28	1591	19	1,2%	977	- 39%	**< 0.01**	106	10,9%
MRBS	24	1595	19	1,2%	837	- 48%	**< 0.01**	54	6,5%
Collabtive	40	2542	72	2,8%	1741	- 31%	**< 0.01**	59	3,4%

WebDriver test suites required less time (from 33% to 57% less) than Sikuli. In one case (i.e., MantisBT), the WebDriver test suite required slightly more time (25% more) to be repaired than the corresponding Sikuli test suite. In another case (i.e., Collabtive), WebDriver required a huge amount of time for test suite repairment with respect to the time required by Sikuli (about 10x more time with WebDriver where, as seen before, we have about 10x more locators to repair). According to the Wilcoxon paired test, the difference in test suite evolution time between Sikuli and WebDriver is statistically significant (at $\alpha = 0.05$) for all test suites (sometimes in opposite directions) except for Claroline (see Table 4). Note that, the maintenance effort is almost entirely due to repair the broken locators (i.e., structural changes) and minimally to modifications to the test cases logic (i.e., logical changes). Indeed, during maintenance, we have approximately modified only the 1% of the LOCs composing the test suites in order to address logical changes of the Web apps. Very often the modifications were exactly the same for both the approaches (i.e., Visual and DOM-based). To summarise, with respect to RQ B.2, the result is not clear-cut. For four out of six considered applications, the effort involved in the evolution of the Sikuli test suites, when a new release of the software is produced, is higher than with WebDriver, but in two cases the opposite is true.

RQ B.3. Table 5 shows data about the time required to execute the test suites. For both tools we report: the mean execution time, computed on 10 replications of each execution; the standard deviation (absolute and relative); the difference in percentage between the time required by the Sikuli test suites and the WebDriver test suites; and, the p value reported by the Wilcoxon paired test, used to compare Sikuli vs. WebDriver's execution times. Execution times range from 1591s to 2774s for Sikuli and from 837s to 1741s for WebDriver. In all the cases, the WebDriver test suites required less time to complete their execution (from -30% to -48%). According to the Wilcoxon paired test, the difference is statistically significant (at $\alpha = 0.05$) for all test suites. To summarise, with respect to the research question RQ B.3 the time required to execute the Sikuli test suites is higher than the execution time of WebDriver for all the six considered applications.

3.6 Qualitative Results

In this section, we discuss on the factors behind the results presented in the previous section, focusing more on the ones that are related to the two approaches and, for space reasons, less on the factors related to the specific tools used:

Web Elements Changing Their State. When a Web element changes its state (e.g., a check box is checked or unchecked, or an input field is emptied or filled), a visual locator must be created for each state, while with the DOM-based approach only one locator is required. This occurred in all the six Sikuli test suites and it is one of the reasons why, in all of them, we have more locators than in the WebDriver test suites (see RQ A.1 and Table 2). As a consequence, more effort both during the development (RQ B.1) and maintenance (RQ B.2) is required in the case of Sikuli test suites (more than one locator had to be created and later repaired for each Web element, RQ A.2). For instance, in MRBS, when we tested the update functionality for the information associated with a room reservation, we had to create two locators for the same check box (corresponding to the slot: Monday from 9:00 to 10:00) to verify that the new state has been saved (e.g., from booked, *checked*, to available, *unchecked*). Similarly, in Collabtive, we had to verify the changes in the check boxes used to update the permissions assigned to the system users.

Changes behind the Scene. Sometimes it could happen that the HTML code is modified without any perceptible impact on how the Web app appears. An extreme example is changing the layout of a Web app from the "deprecated" table-based structure to a div-based structure, without affecting its visual aspect in any respect. In this case, the vast majority of the DOM-based locators (in particular the navigational ones, e.g., XPath) used by DOM-based tools may be broken. On the contrary, this change is almost insignificant for visual test tools. A similar problem occurs when auto-generated ID locators are used (e.g., *id1, id2, id3, ... , idN*) by DOM-based locators. In fact, these tend to change across different releases, while leaving completely unaffected the visual appearance of the Web page (hence, no maintenance is required on the visual test suites). For example, the addition of a new link in a Web page might result in a change of all IDs of the elements following the new link [8]. Such "changes behind the scene" occurred in our empirical study and explain why, in the case of Collabtive, the Sikuli test suite has required by far a lower maintenance effort (see RQ B.2 and Table 4). In detail, across the two considered releases, a minor change has been applied to almost all the HTML pages of Collabtive: an unused div tag has been removed. This little change impacted quite strongly several of the XPath locators (XPath locators were used because IDs were not present) in the WebDriver test suite (see RQ A.2). The majority of the 36 locators (all of them are XPaths) was broken and had to be repaired (an example of repairment is from .../div[2]/... to .../div[1]/...). No change was necessary on the Sikuli visual test suite for this structural change. Overall, in Sikuli, we had only few locators broken. For this reason, there is a large difference in the maintenance effort between the two test suites. A similar change across releases occurred also in MantisBT, although it had a lower impact in this application.

Repeated Web Elements. When in a Web page there are multiple instances of the same kind of Web element (e.g., an input box), creating a visual locator requires more time than creating a DOM-based one. Let us consider a common situation, consisting of a form with multiple, repeated input fields to fill (e.g., multiple lines, each with *Name*, *Surname*, etc.), all of which have the same size, thus appearing identical. In such cases, it is not possible to create a visual locator using only an image of the Web element of our interest (e.g., the repeated *Name* input field), but we have to: (i) include also some context around (e.g., a label as shown in Fig. 4) in order to create an unambiguous locator (i.e., an image that matches only one specific portion of the Web page) or, when this is not easily feasible, (ii) locate directly a unique Web element close to the input field of interest and then move the mouse of a certain amount of pixels, in order to reach the input field. Both solutions locate the target Web element by means of another, easier to locate, element (e.g., a label). This is not straightforward and natural for the test developer (i.e., it requires more effort and time). Actually, both solutions are not quite convenient. Solution (i) requires to create large image locators, including more than one Web element (e.g., the label and the corresponding input field). On the other hand, even if it allows to create a small locator image for only one Web element (e.g., the label), Solution (ii) requires to calculate a distance in pixels (similarly to 1st generation tools), not so simple to determine. Both solutions have problems in case of variation of the relative positions of the elements in the next releases of the application. Thus, this factor has a negative effect on both the development and maintenance of Sikuli test suites. Repeated Web elements occurred in all test suites. For instance, in Claroline, a form contains a set of radio buttons used to select the course type to create. In Sikuli, localisation of these buttons requires either Solution (i) or (ii). Similarly, in AddressBook/MantisBT, when a new entry/user is inserted, a list of input fields, all with the same appearance, has to be filled. In these cases, we created the Sikuli locators as shown in Fig. 4. Recently, JAutomate (http://jautomate.com/), a commercial GUI test automation tool, provided a different solution to this problem by mixing visual locators and position indexes. When a visual locator selects more than one element, it is possible to use an index to select the desired element among the retrieved ones.

Elements with Complex Interaction. Complex Web elements, such as drop-down lists and multilevel drop-down menus, are quite common in modern Web apps. For instance, let us consider a registration form that asks for the nationality of the submitter. Typically, this is implemented using a drop-down list containing a list of countries. A DOM-based tool like WebDriver can provide a command to select directly an element from a drop-down list (only one locator is required). On the contrary, when adopting the visual approach the task is much more complex. Once could, for instance: (1) locate the drop-down list (more precisely the arrow that shows the menu) using an image locator; (2) click on it; (3) if the required list element is not shown, locate and move the scrollbar (e.g., by clicking the arrow); (4) locate the required element using another image locator; and, finally, (5) click on it. All these steps together require more LOCs (in the page objects, see RQ B.1) and locators. Actually, in this case the visual approach performs exactly the same steps that a human tester would do.

Execution Time. The execution time required by the Sikuli tool is always higher than the one required by WebDriver (see RQ B.3 and Table 5). This was expected, since executing an image recognition algorithm requires more computational resources (and thus, generally, more time) than navigating the DOM. However, surprisingly, the difference in percentage between the two approaches is not high, being only 30-48%. It is not very much considering that: (1) Sikuli is a quite experimental tool, (2) it is not focused on Web app testing and, (3) the needed manual management of the pages loading delay (through *sleep* commands) we applied is not optimal[3]. For what concerns the latter point, according to our estimates, the overhead due to the Web page loading delay is not a major penalty for Sikuli (only 20-40 seconds per test suite) as compared to the total processing time. Indeed, we carefully tuned the delays in order to find the smallest required. The standard deviation (see Table 5) is always greater in the case of WebDriver given that, sometimes, it unexpectedly and randomly stops for short periods during test suites execution (e.g., 2-3s between two test cases).

Lesson Learnt: In the following, we report some lessons learnt during the use of the two experimented approaches and tools:

Data-driven Test Cases. Often in the industrial practice [8], to improve the coverage reached by a test suite, test cases are re-executed multiple times using different values. This is very well supported by a programmable testing approach. However, benefits depend on the specific programmable approach that is adopted (e.g., visual vs. DOM-based). For instance, in WebDriver it is possible to use data from various sources, such as CSV files or databases, or even to generate them at runtime. In Sikuli it is necessary to have images of the target Web elements, so even if we can use various data sources (e.g., to fill input fields), when assertions are evaluated, images are still needed to represent the expected data (see Fig. 3). For this reason, in the visual approach it is not possible to create complete data-driven test cases (i.e., for both input and assertions). In fact, while it is indeed possible to parameterise the usage of image locators in the assertions, it is not possible to generate them from data. This happens because using a DOM-based tool there is a clear separation between the locator for a Web element (e.g., an ID value) and the content of that Web element (e.g. the displayed string), so that we can reuse the same locator with different contents (e.g., test assertion values). On the contrary, using a visual tool, the locator for a Web element and the displayed content are the same thing, thus if the content changes, the locator must be also modified. Moreover, it is important to highlight that, if necessary, parameterising the creation of DOM-based locators is usually an easy task (e.g., .//*[@id='list']/tr[**x**]/td[1] with **x**=1..n), while it is infeasible in the visual approach. In our case study, we experienced this limitation of the visual approach since we had, in each test suite, at least one test case that performs multiple, repeated operations that change only in the data values being manipulated, such as: insert/remove multiple different users, projects, addresses, or groups (depending on the considered application). In such cases we used: (1) a single parameterized locator in WebDriver, and (2) several different image locators in Sikuli (e.g., for evaluating the assertions), with the effect that, in the second case, the number of locators required is higher.

Test Case Comprehensibility. The locators used by the two approaches have often a different degree of comprehensibility. For instance, by comparing Fig. 2 with Fig. 4, it is clear that the visual locator pw.png (password) is much easier to understand than the corresponding XPath locator. In fact, the visual approach works in a manner that

[3] A browser needs time to open a Web page. Thus, before starting to perform actions on the page the test automation tool has to wait. WebDriver provides specific commands to deal with this problem (i.e., waiting for the web page loading). In Sikuli this is not available and testers have to insert an explicit delay (e.g., `Thread.sleep(200)`).

is closer to humans than the DOM-based approach. In our case study, we experienced this fact several times. For instance, during test suites maintenance, understanding why a locator is broken is generally easier and faster with Sikuli than with WebDriver.

Test Suites Portability. If a Sikuli test suite is executed on a different machine where the screen resolution or the font properties are different, Sikuli test cases may not work properly. We experienced this problem two times while executing the Sikuli test suites on two different computers: in one case because the default font size was different, resulting in broken image locators, and in another case because the screen resolution was lower than expected, thus more mouse scroll operations were required.

3.7 Threats to Validity

The main threats to validity that affect this study are: Construct (authors' bias), Internal and External validity threats.

Authors' Bias threat concerns the involvement of the authors in manual activities conducted during the empirical study and the influence of the authors' expectations about the empirical study on such activities. In our case, two of the authors developed the test suites and evolved them to match the next major release of each application under test. Since none of the authors was involved in the development of any of the tools assessed in the empirical study, the authors' expectations were in no particular direction for what concerns the performance of the tools. Hence, we think that the authors' involvement in some manual activities does not introduce any specific bias.

Internal Validity threats concern confounding factors that may affect a dependent variable (number of locators, number of broken locators, development, repair, and execution time of the test suites). One such factor is associated with the approach used to produce the test cases (i.e., the chosen functional coverage criterion). Moreover, the variability involved in the selection of the input data and of the locators could have played a role. To mitigate this threat, we have adopted a systematic approach and applied all known good-practices in the construction of programmable test suites. Concerning RQ B.1, learning effects may have occurred between the construction of the test suites for WebDriver and Sikuli. However, this is quite unlikely given the long time (several months) elapsed between the development of WebDriver and Sikuli test suites and the kind of locators (DOM-based vs. visual), which is quite different. Moreover, given the high level of similarity of the test code (in practice, only locators are different), learning would favour Sikuli, which eventually showed lower performance than WebDriver, so if any learning occurred, we expect that without learning the results would be just amplified, but still in the same direction.

External Validity threats are related to the generalisation of results. The selected applications are real open source Web apps belonging to different domains. This makes the context quite realistic, even though further studies with other applications are necessary to confirm or confute the obtained results. In particular, our findings could not hold for RIAs providing sophisticated user interactions, like, for instance, Google Maps or Google Docs. In fact, using a visual approach it is possible to create test cases that are very difficult (if not impossible) to realise with the DOM-based approach. For instance, it is possible to verify that in Google Docs, after clicking the "center" button, a portion of text becomes centred in the page, which is in practice impossible using just the DOM.

The results about number and robustness of locators used by the visual and DOM-based approaches (RQ A.1 and RQ A.2) are not tied to any particular tool, thus we expect they hold whatever tool is chosen in the two categories. On the other hand, the same is not completely true for RQ B.1 and RQ B.2, where the results about the development and maintenance effort are also influenced by the chosen tools, and different results could be obtained with other Web testing frameworks/tools. The problem of the generalisation of the results concerns also RQ B.3 where, for instance, employing a different image recognition algorithm could lead to different execution times.

4 Related Works

We focus our related work discussion considering studies about test suite development and evolution using visual tools; we also consider automatic repairment of test cases.

Several works show that the visual testing automation approach has been recently adopted by the industry [2,6] and governmental institutions [3]. Borjesson and Feldt in [2], evaluate two visual GUI testing tools (Sikuli and a commercial tool) on a real-world, safety-critical industrial software system with the goal of assessing their usability and applicability in an industrial setting. Results show that visual GUI testing tools are applicable to automate acceptance tests for industrial systems with both cost and potentially quality gains over state-of-practice manual testing. Differently from us, they compared two tools both employing the visual approach and did not focus specifically on Web app testing. Moreover, our goal (comparing visual vs. DOM-based locators) is completely different from theirs.

Collins *et al.* [6], present three testing automation strategies applied in three different industrial projects adopting the Scrum agile methodology. The functional GUI test automation tools used in these three projects were respectively: Sikuli, Selenium RC and IDE, and Fitnesse. Capocchi *et al.* [3], propose an approach, based on the DEVSimPy environment and employing both Selenium and Sikuli, aimed at facilitating and speeding up the testing of GUI software. They validated this approach on a real application dealing with medical software.

Chang *et al.* [4] present a small experiment to analyse the long-term reusability of Sikuli test cases. They selected two open-source applications (Capivara and jEdit) and built a test suite for each application (10 test cases for Capivara and 13 test cases for jEdit). Using some subsequent releases of the two selected applications, they evaluated how many test cases turned out to be broken in each release. The lesson drawn from this experiment is: as long as a GUI evolves incrementally a significant number of Sikuli test cases can still be reusable. Differently from us, the authors employed only a visual tool (Sikuli) without executing a direct comparison with other tools.

It is well-known that maintaining automated test cases is expensive and time consuming (costs are more significant for automated than for manual testing [15]), and that often test cases are discarded by software developers due to huge maintenance costs. For this reason, several researchers proposed techniques and tools for automatically repairing test cases. For instance, Mirzaaghaei *et al.* [12] presents TestCareAssistant (TcA), a tool that combines data-flow analysis and program differencing to automatically repair test compilation errors caused by changes in the declaration of method parameters.

Other tools for automatically repairing GUI test cases or reducing their maintenance effort have been presented in the literature [16,7,10]. Choudhary *et al.* [5] extended these proposals to Web apps, presenting a technique able to automatically suggest repairs for Web app test cases.

5 Conclusions and Future Work

We have conducted an empirical study to compare the robustness of visual vs. a DOM-based locators. For six subject applications, two equivalent test suites have been developed respectively in WebDriver and Sikuli. In addition to the robustness variable, we have also investigated: the initial test suite development effort, the test suite evolution cost, and the test suite execution time. Results indicate that DOM-based locators are generally more robust than visual ones and that DOM-based test cases can be developed from scratch at lower cost and most of the times they can be evolved at lower cost. However, on specific Web apps (MantisBT and Collabtive) visual locators were easier to repair, because the visual appearance of those applications remained stable across releases, while their structure changed a lot. DOM-based test cases required a lower execution time (due to the computational demands of image recognition algorithms used by the visual approach), although the difference was not that dramatic. Overall, the choice between DOM-based and visual locators is application-specific and depends quite strongly on the expected structural and visual evolution of the application. Other factors may also affect the testers' decision, such as the availability/unavailability of visual locators for Web elements that are important during testing and the presence of advanced, RIA functionalities which cannot be tested using DOM-based locators. Moreover, visual test cases are definitely easier to understand, which, depending on the skills of the involved testers, might also play a role in the decision.

In our future work we intend to conduct further studies to corroborate our findings. We plan to complete the empirical assessment of the Web testing approaches by considering also tools that implement capture-replay with visual Web element localisation (e.g., JAutomate). Finally, we plan to evaluate tools that combine the two approaches, such as SikuliFirefoxDriver (http://code.google.com/p/sikuli-api/wiki/SikuliWebDriver), that extends WebDriver by adding the Sikuli image search capability, combining in this way the respective strengths.

References

1. Berner, S., Weber, R., Keller, R.: Observations and lessons learned from automated testing. In: Proc. of ICSE 2005, pp. 571–579. IEEE (2005)
2. Borjesson, E., Feldt, R.: Automated system testing using visual GUI testing tools: A comparative study in industry. In: Proc. of ICST 2012, pp. 350–359 (2012)
3. Capocchi, L., Santucci, J.-F., Ville, T.: Software test automation using DEVSimPy environment. In: Proc. of SIGSIM-PADS 2013, pp. 343–348. ACM (2013)
4. Chang, T.-H., Yeh, T., Miller, R.C.: Gui testing using computer vision. In: Proc. of CHI 2010, pp. 1535–1544. ACM (2010)
5. Choudhary, S.R., Zhao, D., Versee, H., Orso, A.: Water: Web application test repair. In: Proc. of ETSE 2011, pp. 24–29. ACM (2011)

6. Collins, E., Dias-Neto, A., de Lucena, V.: Strategies for agile software testing automation: An industrial experience. In: Proc. of COMPSACW 2012, pp. 440–445. IEEE (2012)
7. Grechanik, M., Xie, Q., Fu, C.: Maintaining and evolving GUI-directed test scripts. In: Proc. of ICSE 2009, pp. 408–418. IEEE (2009)
8. Leotta, M., Clerissi, D., Ricca, F., Spadaro, C.: Improving test suites maintainability with the page object pattern: An industrial case study. In: Proc. of 6th Int. Conference on Software Testing, Verification and Validation Workshops, ICSTW 2013, pp. 108–113. IEEE (2013)
9. Leotta, M., Clerissi, D., Ricca, F., Tonella, P.: Capture-replay vs. programmable web testing: An empirical assessment during test case evolution. In: Proc. of 20th Working Conference on Reverse Engineering, WCRE 2013, pp. 272–281. IEEE (2013)
10. Memon, A.M.: Automatically repairing event sequence-based GUI test suites for regression testing. TOSEM, 18(2), 4:1–4:36 (2008)
11. Mirzaaghaei, M.: Automatic test suite evolution. In: Proc. of ESEC/FSE 2011, pp. 396–399. ACM (2011)
12. Mirzaaghaei, M., Pastore, F., Pezze, M.: Automatically repairing test cases for evolving method declarations. In: Proc. of ICSM 2010, pp. 1–5. IEEE (2010)
13. Ricca, F., Tonella, P.: Testing processes of web applications. Ann. Softw. Eng. 14(1-4), 93–114 (2002)
14. Ricca, F., Tonella, P.: Detecting anomaly and failure in web applications. IEEE MultiMedia 13(2), 44–51 (2006)
15. Skoglund, M., Runeson, P.: A case study on regression test suite maintenance in system evolution. In: Proc. of ICSM 2004, pp. 438–442. IEEE (2004)
16. Xie, Q., Grechanik, M., Fu, C.: Rest: A tool for reducing effort in script-based testing. In: Proc. of ICSM 2008, pp. 468–469. IEEE (2008)

Widget Classification with Applications to Web Accessibility

Valentyn Melnyk[1], Vikas Ashok[1], Yury Puzis[2], Andrii Soviak[2], Yevgen Borodin[2], and I.V. Ramakrishnan[2]

[1] Computer Science Department, Stony Brook University, Stony Brook, NY, USA
{vmelnyk,vganjiguntea}@cs.stonybrook.edu
[2] Charmtech Labs LLC, 1500 Stony Brook Rd., Stony Brook, NY, USA
{yury.puzis,and.soviak,borodin,ram}@charmtechlabs.com

Abstract. Once simple and static, many web pages have now evolved into complex web applications. Hundreds of web development libraries are providing ready-to-use dynamic widgets, which can be further customized to fit the needs of individual web application. With such wide selection of widgets and a lack of standardization, dynamic widgets have proven to be an insurmountable problem for blind users who rely on screen readers to make web pages accessible. Screen readers generally do not recognize widgets that dynamically appear on the screen; as a result, blind users either cannot benefit from the convenience of using widgets (e.g., a date picker) or get stuck on inaccessible content (e.g., alert windows). In this paper, we propose a general approach to identifying or classifying *dynamic* widgets with the purpose of "reverse engineering" web applications and improving their accessibility. To demonstrate the feasibility of the approach, we report on the experiments that show how very popular dynamic widgets such as date picker, popup menu, suggestion list, and alert window can be effectively and accurately recognized in live web applications.

Keywords: web applications, reverse engineering, widget classification, widget localization, dynamic widgets, screen reader, web accessibility, ARIA.

1 Introduction

The Web has permeated many aspects of our lives; we use it to obtain and exchange information, shop, pay bills, make travel arrangements, apply for college or employment, connect with others, participate in civic activities, etc. A 2012 report by the Internet World Stats shows that Internet usage has skyrocketed by more than 566% since 2000, to include over a third of the global population in 2012 (over 2.4 billion people) [23]. However, over this time period, the Web has evolved from text-based web pages to interactive web applications, becoming less accessible to blind people.

Many popular websites such as Blackboard, Gmail, Linked-In, Google Drive, eBay, Kayak, YouTube, etc. have turned into sophisticated web applications that utilize dynamic (appearing and disappearing) widgets such as dropdown menus, date pickers, suggestion boxes, etc. to enhance user experience by adding convenient tools.

According to the W3C definition [31], a *widget* (called "widget" thereafter) is defined as a "discrete user interface object with which the user can interact". Widgets can be simple objects including standard HTML controls with a single value or operation (e.g., buttons and textboxes) or they can be complex objects (e.g., trees).

Hundreds of web development libraries and toolkits (e.g., [10, 13, 28] to mention a few) are providing an ever growing number of ready-to-use web widgets that can be further customized by web developers to fit the needs of individual web sites. Unfortunately, the diversity of the libraries and a lack of standardization and enforcement of W3C specifications have proven to be an insurmountable problem for blind users.

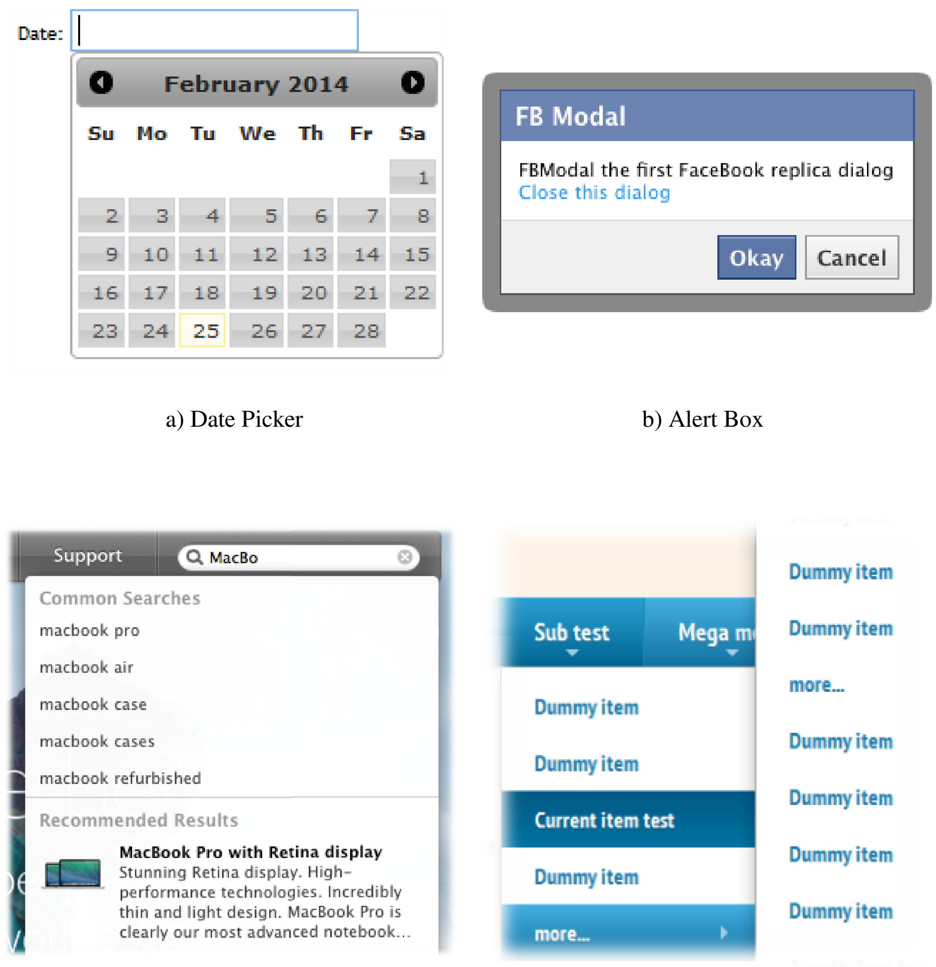
For web browsing, blind people employ screen readers (*e.g.*, JAWS [17], Windows-Eyes [33], VoiceOver [30], Dolphin [27], Sa To Go [26], NVDA [24], etc.), which convert the Web to speech, generally ignoring layout and graphics, and reading aloud all the textual content in web pages. Screen readers enable their users to listen to and navigate web content sequentially in the order it is laid out in the HTML source code, which often does not correspond to visual layout. Screen readers provide many shortcuts to navigate among elements of a particular type, e.g., links, buttons, edit fields, etc. Although not very efficient for web browsing [3], screen readers enable visually-impaired people to browse the Web and perform online activities.

Unfortunately, screen readers do not recognize widgets that dynamically appear on the screen, so the user has no easy way to find them; at best, a dynamically appearing widget will be "navigable," meaning that it can be found and narrated by the screen reader, but without giving any indication as to what kind of widget it is. As a result, blind users either cannot benefit from the convenience of using widgets (e.g., use a date picker) or they even get stuck on inaccessible content (e.g., an HTML alert window). So, while sighted people can enjoy Rich Internet Applications (RIA), blind people either cannot access them at all (e.g., cannot use Google Docs) or have to use basic versions of the websites (Gmail and Facebook).

To make web applications more accessible, web developers have to follow Accessible Rich Internet Applications (ARIA) specifications [32]. For instance, ARIA allows developers to mark up live regions where the content may update, specify the importance of those updates, and provide simple roles such as "progress meter." Unfortunately, web developers do not follow ARIA specifications consistently, and ARIA does not have predefined roles for complex widgets such as date picker.

In this paper, we propose an ARIA-independent approach towards improving the accessibility of dynamic widgets. Specifically, the contribution of this paper is a scalable machine learning approach to identification/classification of *dynamic* widgets. To demonstrate the feasibility of the approach, we demonstrate high accuracy in classifying popular dynamic widgets such as date pickers, popup menus, suggestion lists, and HTML alert windows. Sample screenshots of these widgets are shown in Fig. 1.

The identification of *static* widgets (the ones that are already in the web page) and the design and evaluation of accessible user interfaces for widgets are beyond the scope of this paper, as these have been well explored in the literature, e.g., [7, 29].



FBModal the first FaceBook replica dialog

MacBook Pro with Retina display
Stunning Retina display. High-performance technologies. Incredibly thin and light design. MacBook Pro is clearly our most advanced notebook...

a) Date Picker b) Alert Box

c) Suggestion Box d) Popup Menu

Fig. 1. Sample screen shots of the four widget types collected in the corpus: a) Date Picker, b) Alert window, c) Suggestion Box, and d) Popup menu

In the remainder of this paper, we provide some background on the complexities of dealing with dynamic widgets as well as on the specifics of using ARIA in Section 2; we review prior work relevant to widget localization and classification in Section 3; we describe the setup of the experiments in Section 4; we present and discuss the obtained results in Section 5, and, finally, we conclude by providing the broader impact view of this work and propose future directions of research in Section 6.

2 Background

In order to help understand the technology behind dynamic widgets, and thus the complexity of identifying them, we give a short overview of a variety of methods employed for making the widgets "appear" and "disappear" in web pages. We also provide background on how ARIA [32] can be used to mark up widgets in a way that would make them accessible with screen readers.

2.1 Methods for Displaying Dynamic Widgets

The most straightforward approach to displaying a new widget on the screen is to use JavaScript to insert a new sub-tree representing the widget into the HTML DOM tree. However, in order for this newly inserted widget, or any other widget that is already a part of the DOM tree to appear on the screen, the web developer must set a long list of widget's properties. Modifying any one of them can make the widget invisible or visible for the user. The following is a non-exhaustive list of "tricks" that web developers use to make the widget "appear" or "disappear" (this list was collected from HTML specification [16], numerous developer forums, and confirmed by observing the behavior of hundreds of widgets in the wild):

- Coordinates can be set to a negative value (off-screen) to hide a widget, and set to a positive value to show a widget within the webpage viewport;
- Coordinates can be set to place a widget within or outside of the viewport of a containing object, which can also be another widget;
- Dimensions of a containing object's viewport can be set in a way that the contained widget is hidden or shown on the screen;
- The width and height of a widget can be set to a value that would make it big enough to be visible or make it so small that it disappears;
- The widget's color can be set to blend in with the background or make it stand out;
- The font can be set to a human-readable size or to 0 to make the textual content of widgets appear or disappear;
- The opacity of the widget can be set to make it transparent or opaque;
- The "visibility" or "display" style of the widget can be set as desired;
- The Z-index of the widget can be set to place it behind or in front of other objects.

All these properties can be modified directly in the HTML code of the widget or indirectly, e.g., by changing the widget's "style" attribute. Further, some of these properties (e.g., "display") are inherited from ancestor DOM nodes, so adding a widget as a sub-tree to the DOM will immediately result in the application of ancestor's properties unless they are overridden.

The code and content of widgets can be in the HTML, Cascading Style Sheets (CSS), in JavaScript, or it can be delivered to the webpage via AJAX from the web server. Given the variety of ways to hide and show a widget and deliver it to the web page, the only good way to detect the appearance of dynamic widgets is by monitoring the changes in the DOM tree [12], as we describe in Section 4.1.

2.2 Screen Readers and Accessible Rich Internet Applications (ARIA)

As the blind user is “navigating” on a webpage, screen readers maintain the virtual cursor pointing at the currently narrated object. When dealing with widgets, screen-readers have three main tasks: 1) allow the user to move a virtual cursor from widget to widget, and from one atomic object on the webpage to another; 2) announce relevant information at the current (new) virtual cursor position, including: textual content of the widget, semantics or purpose of the visited widget (e.g., text box, alert window, etc.), its state (e.g., “blank”, “disabled”, etc.), and the results of user actions affecting the widget (e.g., typed letter, selected item, etc.), and 3) announce important changes to the webpage not localized at the current virtual cursor position.

Unfortunately, the best screen readers can only accomplish the tasks above if widgets are marked up following the Accessible Rich Internet Applications standard (ARIA) [32]. And the responsibility for making web content accessible lies entirely on web developers most of whom take the off-the-shelf ready-to-use widgets from a widget library. Alas, ARIA is not widely followed by web developers or even by screen reader developers. Furthermore, due to its complexity, it is often incorrectly implemented by web developers, which makes the accessibility problem even worse.

Making a Widget Accessible with ARIA. ARIA markup is not necessary for making standard widgets (e.g., textbox, link, listbox, button, etc.) accessible, because screen readers already have a long list of sophisticated hardcoded interaction rules that enable: a) virtual cursor’s movement between widgets, b) user interaction with the widgets, and c) relevant announcements that are made to the user when the cursor arrives or leaves the widget or changes its value.

For custom widgets which model the standard widgets, web developers can use the ARIA roles, states, and properties that enable the screen reader to map the custom widget to a standard widget and activate corresponding hardcoded rules. For example, a checked checkbox implemented using JavaScript and html tag “div” can be marked up with attributes role=“checkbox” and aria-checked=“true”. When the checkbox is unchecked, the author needs to use JavaScript to set aria-checked=“false”.

For custom widgets that model a combination of standard widgets web authors can use a complex combination of ARIA roles, states and properties. ARIA authoring practices can be found in [19], and a good overview of accessible widgets in [21].

For custom widgets that, due to their functional complexity or uniqueness, cannot be mapped to any standard widget (e.g., calendar), web developers need to: a) inform the screen-reader that this is a custom widget without explicit mapping to a standard widget by specifying the widget’s role as “application”; b) process relevant keyboard commands, including those that would normally be handled by the screen-reader (e.g., arrow keys), and map those keys to the widget’s functionality; c) if the widget consists of multiple interactive components (e.g., a grid with editable grid cells), the web author needs to assign some shortcuts to move the browser’s focus between all those components (this will hint the screen-reader to follow the focus with the virtual cursor, and narrate the relevant content).

To announce changes to the webpage not localized at the current virtual cursor position the authors can use ARIA Live Regions; designated by attribute “live”, it can be set to one of four politeness levels (“off”, “polite”, “assertive”, “rude”) determining the urgency (importance) of the changes in the region. The attribute “relevant” is used to set the relevance of specific types of DOM changes within the live region.

The Problems with Using ARIA. Using standard widgets or a custom implementation of standard widgets enables the screen-reader to implement a powerful, accessible interface to a very limited set of widgets. This requires no or little overhead for the web developer, but, in case of custom widgets, this often depends on the diligence in assigning roles and dynamically modifying attributes as a result of user interactions.

Using a complex combination of ARIA roles and states to implement a single widget has the potential to expand the range of functionality and types of widgets supported by the screen-readers. However, this approach requires advanced understanding of ARIA by the developer, increases cost of implementation and support, depends on advanced support of ARIA by screen-readers, which is not yet available for the full ARIA specification, and is limited by the scope of ARIA specifications that do not cover all possible types of functionality. As a result, the use of ARIA in complex widgets (e.g., date picker, etc.) to make them accessible is not widely used.

The use of the “application” role in combination with JavaScript code for processing keyboard events and controlling browser focus has the potential to enable even more complicated widgets than the other approaches. However, predictably, this requires even more effort on the part of the developers than the purely ARIA approach, and, hence, this approach is not widely deployed either.

Issues with ARIA live regions, including difficulties with determining causality of the region updates, giving developers the ability to combine technically discrete but semantically atomic updates, handling interim updates, and providing higher-level abstractions for web developers, are well summarized in [29].

Overcoming the Widget Accessibility Problems. In order to make a widget accessible, it needs to be first localized and identified. Once localized, the widget needs to be enabled for screen reader interaction which can be accomplished either by injecting ARIA into the widget components [9, 18] or by enabling the screen-reader to map the widget (or its components) to the hardcoded interaction rules that will make this widget accessible, similar to the way it is done with the standard widgets. The latter approach is both more powerful than ARIA injection (because it is not restricted by the expressively of the ARIA specifications) and is less prone to errors since it is not restricted by the ability of other screen-readers to correctly interpret ARIA.

In this paper, we focus on the classification or identification of dynamic widgets, which is the first step toward making them accessible. Customizing and evaluating user interfaces for widgets [7, 29] is not in scope of this paper. However, it is notable that the proposed method does not limit the way widgets will be made accessible.

3 Related Work

Many researchers have recognized the accessibility problems caused by dynamic content and dynamic widgets (those that dynamically appear and disappear on the screen), as well as the deficiency of screen readers in handling this problem [1, 2, 4, 6, 8, 9, 11, 14, 18, 20]. Several approaches were proposed to make dynamic content and widgets accessible, localize widgets, and classify the widgets into types.

3.1 Making Dynamic Widgets Accessible

The majority of the approaches described in the literature focused primarily on retrofitting the existing web application by adding ARIA markup [32] to the web page, while a few attempted to provide a custom screen-reader interface.

An early approach proposed by the authors [4] enabled users to review any dynamic updates on the web page using a special layered view isolating the changes. While useful, that approach, however, did not help identify the type of changes or their importance; and, without the ability to isolate and classify the type of widgets, it is impossible to provide a usable interface that works best with a particular widget.

For example, if a slideshow widget changes, unless the screen-reader user is on it, it is a low priority event that should not interrupt what the user is listening to. On the other hand, if an alert window appears, the event has to be audibly announced, screen-reader navigation should only be available within the scope of the window, and the text of the window needs to be read out. If a date picker appears, current date has to be announced, and table navigation (left/right/up/down) has to be enabled in the calendar.

AxsJax [9] was one of the pioneering approaches that showed (using the example of Google chat) how web developers can make their web applications more accessible by injecting ARIA metadata into AJAX (Asynchronous JavaScript) responses sent from the server to the web browser. The injection would happen on the server-side and would have to be enabled by the web developers. However, if the developers were motivated to make their websites accessible, they would have ARIA from the start. So, AxsJax, does not fix the accessibility problem, but rather provides web developers with an alternative method for making their web applications accessible.

Single Structured Accessibility Stream for Web 2.0 Access Technologies (SASWAT) [18] is a project performing the AJAX injection on a proxy server. This releases the web developers of responsibility and puts it into the hands of the SASWAT supporters, who would have to provide the ARIA metadata describing the dynamic content and store it into a repository. Then, if any of the specified webpages were loaded through the proxy, the proxy would inject the ARIA metadata into the webpages, thus making them accessible. Of course, this approach breaks down if the web pages are changed by the developers and it is not scalable across websites, because any variation of a widget would require modifications of the scripts. A truly scalable approach would require automatic widget localization and classification as we discuss in the following section.

3.2 Widget Localization and Classification

In [1], authors propose an approach to widget security vulnerability detection. For this purpose, the authors locate widgets using DOM dynamic updates analyses. They track the parts of the DOM, which were changed as a result of some user action or JavaScript event. Later, they test identified widgets for inter change, i.e. when one widget is updated automatically by another widget. While there is some similarity in tracking mutation events used in this paper, authors do not care about the types of widgets.

The approach proposed in [2] utilizes end-user-programming to enable automation and customization of web application. Based on the selected keywords, one can build a pattern to localize a widget on the web page, which could potentially be reused across websites. However, this approach only works for simple widgets (search box or a button) and requires manual effort for selected keywords for a particular widget.

Several research teams have looked into the possibility of recognizing complex dynamic widgets. For instance, [7] has explored the possibility of detecting calendar widgets; however, the approach was a special-case heuristic classification that detected the calendar based on user interaction, which is not scalable to other widget types.

An early attempt to classify different widgets was made in [8]. The proposed algorithm analyzed webpage sources (HTML, JS, and CSS) using regular expressions to find the display window (the area containing the widget) and then matching the content to an ontology with predefined widget features. Unfortunately, the classification was done by constructing a hierarchy of widgets based on widget implementations in specific libraries, which means that the classification will work only for the widgets taken from these libraries. Also, the ontology has to be created manually, and the approach will not be able to detect the (dis)appearance of a dynamic widget.

A more dynamic approach to desktop widget classification is proposed in [11]. The main goal of the paper is to simplify a desktop interface and make it "visually accessible". The system is built as an extension of the Prefab pixel-based recognition system. The system recognizes some simple widgets such as "Windows 7 steal buttons" based on the visual markers. Besides requiring compute-intensive vision based analysis of the screen, unfortunately, this approach is also limited by the visual distinction between widgets. For example, it will not able to distinguish a suggestion box from the popup menu because both look like popups. Neither will it be able to find a widget if it is hidden behind some other control such as a drop down menu.

In contrast, the method proposed in this paper overcomes the limitations listed above by using a more general and lightweight approach to widget classification. It employs machine learning to classify widgets automatically regardless of the library they come from. Since a variety of features can be extracted from the DOM tree (Section 4.2), the approach can classify widgets even if they are visually similar, e.g., based on trigger events. Furthermore, once the widgets are classified, this information can be used either to inject ARIA metadata into the webpage or provide this information directly to the screen reader to provide customized interaction with the widget.

The limitation of the proposed approach is that it can only identify *dynamic* widgets that appear and disappear in web pages. However, some of the reviewed methods [2, 8, 11] can be employed to detect *static* widgets that are already on the web page.

4 Experimental Setup

4.1 The Corpus and Widget Localization

To experiment with widgets, we collected a corpus with four types of popular widgets (suggestion list, HTML alert window, popup window, and date picker, shown in Fig. 1) with 50 examples of each widget type. To this corpus, we added an additional 50 examples of other randomly selected widget types; we refer to this generic widget type as "others". The corpus was collected from widget libraries and live websites using custom tool developed specifically for this purpose.

The data collection tool, based on the Capti Narrator (www.captivoice.com) for Mac/Windows [5], consists of a Firefox browser extension and a Java application. The browser extension listens to all DOM mutation events (updates), and communicates them over an open socket to the Java application. Java application uses the updates to construct a timeline of DOM mutations, reconstructs the DOM at any given point of time, and displays it in a separate window, in the form of a tree. Once a website with a widget has loaded, the process of collecting the data of that single widget is semi-automatic:

1. Press a button control shortcut to start "recording" all DOM mutation events;
2. Trigger opening of the widget and wait for it to open (usually instantaneous), e.g., press a button opening an alert window, focus on a textbox with a date picker, etc.;
3. Press the same button to terminate the "recording" of DOM mutations;
4. Verify that the recorded DOM mutations represent the target widget;
5. Save the resulting timeline of DOM mutations into the corpus of widgets.

The recorded timeline spans the period of "recording" and includes only the events in two sub-trees: one representing the trigger object and the other representing the widget that opened as a result of the trigger; all other events are ignored. The data collection tool was designed with the following considerations in mind.

A webpage may have many scheduled DOM mutation events, so it is very important to localize the relevant DOM mutations. While JavaScript is executed synchronously in most browsers, the exact localization of the mutation events relevant to a particular widget is an unsolved problem; multiple unrelated mutation events can happen immediately one after another. So, automated analysis of the underlying JavaScript would be required to understand the relationship between the user action (e.g., pressing a shortcut) and the system reaction (e.g., displaying a widget).

However, in practice, a heuristic approach that uses both temporal and spatial information helps minimize the risk of collecting irrelevant mutation events. Specifically, any user event such as focus change or control activation can be considered to be the potential trigger event, starting an observation period. Any subsequent DOM mutation events occurring within time *t* of the trigger event can be considered candidates for the widget. If more than one DOM sub-tree has updated, the collected mutation events can be further filtered by their spatial proximity to the trigger object.

4.2 Features for Widget Classification

The selection of features was inspired by the observations made during a manual inspection of the corpus. The vector representation of features was assembled from the features extracted from four different categories listed in Table 1. Most of the examined features are binary with the exception of the "Proportion of text nodes with only numbers in them" and the "Number of text nodes".

Table 1. Feature space for widget classification

Feature	Description	Binary
	PRESENCE OF HTML TAGS & KEYWORDS	
P_{table}	Presence of table tag <table> in the HTML associated with widget	Yes
P_{list}	Presence of list related tags like <ul>, <li>, etc, in the HTML associated with widget	Yes
$P_{textbox}$	Presence of textboxes (e.g. <input type= "text">) in the HTML associated with the widget	Yes
P_{name}	Presence of widget name in "class" attribute of any tag in the HTML associated with the widget	Yes
P_{date}	Presence of "date" as the value of "type" attribute in any tag in the HTML associated with the widget	Yes
P_{image}	Presence of an image (<img>) in the HTML associated with the widget	Yes
	CONTEXT RELATED	
T_{link}	Widget appears due to click of a button or link	Yes
T_{input}	Widget appears due to a keyboard entry in an input box	Yes
	CHARACTERISTICS OF NODES IN WIDGET DOM SUBTREE	
$C_{text.num}$	Number of text(<text>) nodes	No
$C_{text.prop}$	Proportion of text (<text>) nodes containing only numbers in them	No
$C_{table.list}$	A table (<table>) or list (<ul>) is present and over 80% of its content are links	Yes
	DISPLAY PROPERTIES OF WIDGET	
D_{widget}	Widget appears right below the "triggering" element	Yes

Presence of HTML Tags and Keywords (P). Analysis of the corpus revealed that, in some widgets, certain HTML elements are almost always present. For example, a list of links (<ul><li>…</li></ul>) can be found with high probability in a popup menu, a table (<table>) is likely to be present in a date picker widget to format the calendar, and an input textbox is always a part of suggestion list. In addition to HTML tags, attribute values can also be used to identify the widget. For example, we observed that, in many cases, the "class" attribute of one of the <div> or <span> HTML tags contained the name of the widget (e.g., "suggestion-box", "date-picker", etc.).

Context-Related Features (T). The local context surrounding a widget provides valuable information and important cues for identifying that widget. We refer to any HTML element that causes the appearance of a widget as a "trigger". The fact that different widgets are triggered in different ways by different HTML elements can be exploited to improve widget classification performance. For example, a suggestion box is almost always triggered when a user types something in an input text box; a menu popup appears on-screen when a user presses the corresponding button; a date picker widget appears when the user goes in focus on the textbox, etc.

Characteristics of Nodes in Widget DOM Sub-tree (C). This set of features was crafted after an extensive analysis of the DOM sub-trees corresponding to different widgets in the corpus. These features strive to leverage differences in the composition of DOM sub-trees corresponding to different widgets. For example, we observed that the DOM sub-tree of a suggestion box contains relatively higher number of <text> nodes compared to the DOM sub-tree of an alert box. Variation in composition also includes the type of content stored in the DOM nodes. For example, a list in a menu pop-up widget is likely to contain a high percentage of links, whereas a list in a suggestion box widget is likely to contain a large number of text nodes.

Display Characteristics of Widget (D). The position of a widget on the screen provides an important cue for its classification. Specifically, we are interested in the screen location of the widget relative to its triggering HTML element. This is based on our observation that different widgets exhibit different display patterns relative to their corresponding trigger. For example, in our corpus, the suggestion box widget appeared right below the triggering input textbox 100% of the time, and the menu pop widget always appeared either below or to the right of the corresponding triggering button or link. This feature is especially useful to filter out "irrelevant" dynamic mutations that can happen in the same time frame when the widget appears.

Having identified the salient features that could help distinguish different widgets, we conducted experiments to identify which combination of these features with which machine learning tools would yield the best widget classification results.

5 Experiments and Results

To conduct the experiments, we used the Weka [15] toolkit. We considered several popular machine learning classifiers and selected the following classifiers that yielded the best performances: Support Vector Machine, frequently used for benchmarking, and the J48 Decision Tree classifiers, which is a simple rule-based classifier that is appropriate for the mostly binary widget features.

As described in the previous section, we divided the features into five thematic categories according to the type of the feature. In order not to evaluate each feature separately, we combined the categories into five groups (Groups 1-5 in Table 2) in a way that would allow us to evaluate the impact of each individual category on the results. For example, Presence (P) features are absent in Group 1, Context (T) features are absent in Group 2 and so on. Group 5 has all the feature sets, and hence the performance of group 1-4 can be compared with Group 5 to assess the importance of the corresponding missing feature set.

Table 2. Feature groups used for widget classification

Group	Features
Group 1	Context (T) + Characteristics (C) + Display (D)
Group 2	Presence (P) + Characteristics (C) + Display (D)
Group 3	Presence (P) + Context (T) + Display (D)
Group 4	Presence (P) + Context (T) + Characteristics (C)
Group 5	Presence (P) + Context (T) + Characteristics (C) + Display (D)

Finally, we ran both classifiers on each of the five groups of features. We used 5-fold cross validation: 200 widget examples (40 of each type) were used for training and 50 (10 of each type) for testing, repeated 5 times with different divisions into folds. The results of the experiments are detailed in Table 3; the winning group-widget type combinations are in bold.

The absolute winners (J48 on Groups 4 and 5) have shaded background. As can be seen from the averages in Table 3, SVM and J48 classifiers yielded similar performance: SVM showed highest performance: 86% recall and precision in groups 4 and 5, while J48 won by a single percent point in both precision and recall. In all groups, J48 was performing better than SVM on average. In general, SVM yielded slightly better performance than J48 in identifying Date Pickers, but J48 was better at distinguishing Alert Boxes and Popup Menus. The absolute best performance was shown by J48 in Groups 4 and 5, yielding precision: 87% and recall: 87%, beating the SVM by 1% in both precision and recall.

Group 5 yielded the best average performance (Precision: 91%, Recall: 94%) when the generic widget type "*Others*" was excluded from the analysis, beating Group 4 (Precision: 90%, Recall: 94%) by a narrow margin of 1% in precision. In all of the feature groups, the average precision excluding "*Others*" is only slightly better than average precision with "*Others*". However, the average recall excluding "*Others*" is significantly better than the overall average recall, in all of the feature groups. These performance results demonstrate the effectiveness of our models in accurately identifying the 4 core widget types considered in our work.

Table 3. Widget classification results, Notation: P - Precision, R - Recall, J48 - Decision Tree, SVM - Support Vector Machine; the values in bold indicate the best performances per group

Widget	Classifier	Group 1		Group 2		Group 3		Group 4		Group 5	
		P	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**
Suggestion box	SVM	**0.96**	**0.96**	0.81	0.88	0.76	0.68	**0.96**	**0.96**	**0.96**	**0.96**
	J48	**0.96**	**0.96**	0.80	0.96	0.94	0.58	**0.96**	**0.96**	**0.96**	**0.96**
Alert Box	SVM	0.65	0.76	0.73	0.90	0.67	0.84	0.74	0.94	0.77	0.94
	J48	0.92	0.78	0.84	0.78	0.76	0.84	**0.83**	**0.90**	0.85	0.88
Menu Popup	SVM	0.47	0.68	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82
	J48	0.61	0.62	0.79	0.88	0.80	0.80	0.83	0.90	**0.85**	**0.90**
Date Picker	SVM	**0.98**	**1.00**	**0.98**	**1.00**	0.61	0.64	**0.98**	**1.00**	0.97	1.00
	J48	0.96	1.00	0.96	1.00	0.61	0.82	0.96	1.00	0.96	1.00
Others	SVM	0.70	0.24	0.59	0.36	0.64	0.44	0.79	0.54	0.80	0.56
	J48	0.59	0.60	0.58	0.42	0.62	0.54	**0.78**	**0.60**	0.74	0.60
Overall Avg.	SVM	0.75	0.73	0.78	0.79	0.70	0.68	0.86	0.85	0.86	0.86
	J48	0.81	0.79	0.79	0.81	0.75	0.72	**0.87**	**0.87**	**0.87**	**0.87**
Avg. excl. Others	SVM	0.77	0.85	0.83	0.90	0.72	0.75	0.88	0.93	0.88	0.93
	J48	0.87	0.84	0.85	0.91	0.78	0.76	0.90	0.94	**0.91**	**0.94**

Group 3, missing the Characteristics (C) features had the worst results both with the SVM (average P: 70%, R: 68%) and J48 (average P: 75%, R: 72%) classifiers. This shows that Characteristics (C) features were very important for classification of widgets in general. The results of both Group 1 and Group 2 are also significantly worse than Group 5, thereby demonstrating the importance of Presence (P) and Context (T) features. However, the absence of Display (D) feature (Group 4) did not cause any significant drop in performance (compared to Group 5 containing D feature). Overall, it can be inferred from Table 3 that feature sets P, T and C are critical for high performance, while the feature set D has minimal impact on the performance.

Notice in Table 3 that the accuracy in identifying Suggestion Boxes is heavily dependent on the T and C features (there is a significant drop in performance in Groups 2 and 3 compared to Group 5), while the performance is least influenced by the P and D features (Groups 1 and 4 yield the same performance as Group 5). Similarly, in case of Date Pickers, the C features were seen to contribute the most towards performance improvement compared to P, T and D features.

Table 4 lists the top discriminatory features for each widget as determined by the SVM classifier. As expected, T_{input} is the topmost predictive feature for Suggestion Box widget since all suggestion boxes are always activated when a user types something in an input textbox (often a search box). Also, observe that the feature P_{image} is also highly predictive of Suggestion Boxes, which indicates that a lot of websites provide suggestion boxes that contain image icons in addition to the suggested links or text. An example of such a suggestion box is shown in Fig. 1(c).

Table 4. Top discriminatory features as determined by the SVM classifier weights. According to [25], high positive weights indicate high predictiability of the corresponding class.

Widget	Top Discriminatory Features
Suggestion Box	T_{input}, P_{list}, P_{image}, $C_{table.list}$
Alert Box	P_{image}, T_{input}, $P_{textbox}$
Menu Popup	P_{list}, P_{name}, $C_{table.list}$
Date Picker	$C_{text.prop}$, P_{table}, T_{input}, P_{name}, $C_{text.num}$, $C_{table.list}$

It can be also seen in Table 4 that P_{name} is predictive of Menu Popup and Date Picker widgets, but not the other two widgets, thereby, highlighting a difference in the way these types of widgets are implemented with reference to CSS; compared to Alert windows and Suggestion boxes, a higher percentage of Popup Menu and Date Picker widgets have at least one node in their DOM sub-trees storing the corresponding widget name as a class attribute. Also, as expected, P_{list} is a top discriminatory feature for Menu Popup, since almost all pop-up menus contain a list of selectable items.

It can be inferred from Table 4 that overall, the features related to the DOM subtree characteristics (C) are extremely useful for widget classification.. This claim is supported by two observations: (i) Feature Group 3 yielded the lowest performance among the feature groups as previously noted in Table 3 analysis, and (ii) 3 out of 4 classes in Table 4 have at least one top discriminatory feature belonging to this category, e.g., $C_{text.num}$ points to variations in textual composition of different widgets.

Figure 2 depicts the decision tree (considering the entire dataset and all the features) produced by the J48 decision tree algorithm supported by WEKA toolkit. It can be inferred from Figure 2 that the feature $C_{text.prop}$ is the most important feature for identifying Date Picker widget type. More specifically, the proportion of text nodes with only numbers, in all Date Picker widgets in the dataset was above 0.21. Only one other widget of a different type in the dataset had the value of $C_{text.prop}$ greater than 0.21 (The label ‘51/1’ of Date Picker leaf node in Figure 2 indicates that out of 51 data points placed in that group, 1 of them is incorrectly classified).

Similar inferences can be made from the decision tree in Figure 2 with respect to other widget types. For example, it can be observed that the context feature T_{input} is critical for correctly identifying the Suggestion Box widget type. Recall that even the SVM classifier determined T_{input} to be the most discriminating feature for identifying the Suggestion Box widget type (Table 4).

Similarly, it can be seen that P_{list} is the most important feature required for the accurate classification of Menu Popup widget type. Also observe in Figure 2 that the feature P_{name} plays an important role in identifying those Menu Popup data points that are not covered by P_{list}. These observations are in accordance with the SVM results presented in Table 4 where P_{list} and P_{name} are the top two discriminating features for Menu Popup widget type. However, no such straightforward comparisons between SVM and J48 decision tree results can be made with respect to the Alert box widget type as it can be observed in Figure 2 that the Alert Box widget type relies on different combinations of a wide variety of features for their accurate identification.

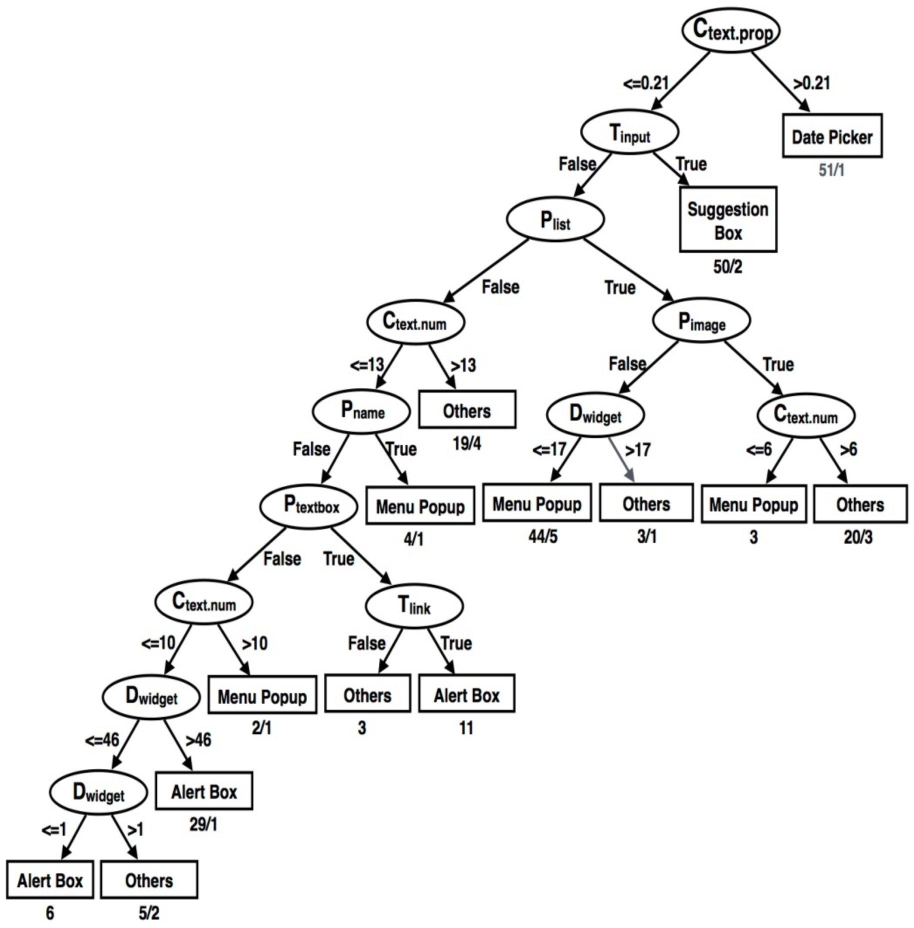


Fig. 2. Decision tree produced by the J48 algorithm on the entire dataset for feature group 5. Each leaf label indicates total classified data points followed by total incorrect classifications.

The importance of each group of features is apparent from the decision tree in Figure 2, where all features except P_{table} and P_{date} are used for classification. The absence of P_{table} in Figure 2 is a bit surprising since P_{table} was determined to be one of the top discriminatory features for Date Picker widget type. This observation adds weight to our earlier deduction that $C_{text.prop}$ is the single most important feature for identifying Date Picker widgets.

6 Conclusion and Future Work

In this paper, we have proposed and evaluated a scalable method for classification of dynamic web widgets using machine learning. The experiments on a corpus of 250 widgets showed that the decision tree learning was the most accurate machine learning technique for identifying and distinguishing among four popular types of widgets: the popup menu, the HTML alert window, suggestion box, and data picker (Fig. 1).

To date, there exists no assistive technology capable of enabling consistent and usable interaction with dynamic content and dynamic widgets. As was discussed in Sections 2 and 3, the existing solutions are very limited and are inadequate given the continuing and rapid adoption of dynamic web technologies. The ability to access the same web applications that are available to sighted people will enable people with vision impairments to enjoy the latest assistive technology, making them more productive. The approach proposed in this paper is the first step towards making widgets accessible, requiring further research and development in this direction.

While this paper handled four popular types of dynamic widgets, there are many more types of dynamic widgets that are used more rarely, but are used nonetheless. Although the proposed approach is scalable for more types of widgets, an extensive dataset has to be first assembled in order to handle more types of widgets. New approaches to dynamic widget localization have to be explored and tested. A reliable method needs to be developed for identifying static widgets that are already in the web page as soon as it loads.

In parallel, automatically identified widgets need to be made accessible to screen readers. While injecting ARIA (Section 3.1) may provide general accessibility to all screen readers supporting ARIA, it is also possible to embed the method described in this paper into a screen reader. The latter approach will allow for a more powerful user interface that is not possible given the limited expressivity of ARIA. Longitudinal user studies will have to be conducted to evaluate the usability of the user interfaces and verify the accuracy of the widget identification approach in the wild.

Finally, the approach proposed in this paper can find use in other web-based application areas. For instance, widget identification can be useful in website crawling [22], website simplification [11], and other reverse engineering of web applications.

Acknowledgements. This work was developed under a grant from the Department of Education, NIDRR grant number H133S130028. However, contents do not represent the policy of the Department of Education, and you should not assume endorsement by the Federal Government. We are also grateful to our Accessibility Consultant, Glenn Dausch, for his insightful feedback on accessibility problems with dynamic widgets.

References

[1] Bezemer, C.-P., Mesbah, A., Deursen, A.V.: Automated security testing of web widget interactions. In: Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, pp. 81–90. ACM, Amsterdam (2009)

[2] Bolin, M., Webber, M., Rha, P., Wilson, T., Miller, R.C.: Automation and customization of rendered web pages. In: Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology, pp. 163–172. ACM, Seattle (2005)

[3] Borodin, Y., Bigham, J.P., Dausch, G., Ramakrishnan, I.V.: More than meets the eye: a survey of screen-reader browsing strategies. In: Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A), pp. 1–10. ACM, Raleigh (2010)

[4] Borodin, Y., Bigham, J.P., Raman, R., Ramakrishnan, I.V.: What's new?: making web page updates accessible. In: Proceedings of the 10th International ACM SIGACCESS Conference on Computers and Accessibility. ACM, Halifax (2008)

[5] Borodin, Y., Sovyak, A., Dimitriyadi, A., Puzis, Y., Melnyk, V., Ahmed, F., Dausch, G., Ramakrishnan, I.V.: Universal and ubiquitous web access with Capti. In: Proceedings of the International Cross-Disciplinary Conference on Web Accessibility, pp. 1–2. ACM, Lyon (2012)

[6] Brown, A., Jay, C., Chen, A.Q., Harper, S.: The uptake of Web 2.0 technologies, and its impact on visually disabled users. Univers. Access Inf. Soc. 11(2), 185–199 (2012)

[7] Brown, A., Jay, C., Harper, S.: Audio access to calendars. In: Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A), pp. 1–10. ACM, Raleigh (2010)

[8] Chen, A., Harper, S., Lunn, D., Brown, A.: Widget Identification: A High-Level Approach to Accessibility. World Wide Web 16(1), 73–89 (2013)

[9] Chen, C.L., Raman, T.V.: AxsJAX: a talking translation bot using Google IM: bringing Web-2.0 applications to life. In: Proceedings of the 2008 International Cross-Disciplinary Conference on Web Accessibility (W4A). ACM, Beijing (2008)

[10] DevExpress. DevExpress Widget Library (2014), `https://www.devexpress.com` (cited 2014)

[11] Dixon, M., Leventhal, D., Fogarty, J.: Content and hierarchy in pixel-based methods for reverse engineering interface structure. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 969–978. ACM, Vancouver (2011)

[12] DOM. W3C Document Object Model (2004), `http://www.w3.org/DOM/DOMTR` (cited 2010)

[13] Google. Google Web Toolkit (2014), `http://gwt-ext.com/demo/` (cited 2014)

[14] Hailpern, J., Guarino-Reid, L., Boardman, R., Annam, S.: Web 2.0: blind to an accessible new world. In: Proceedings of the 18th International Conference on World Wide Web, pp. 821–830. ACM, Madrid (2009)

[15] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update. SIGKDD Explorations (2009)

[16] HTML5. Hyper-Text Markup Language v.5.0 (2010), `http://dev.w3.org/html5/spec/` (cited 2010)

[17] JAWS. Screen reader from Freedom Scientific (2013), `http://www.freedomscientific.com/products/fs/jaws-product-page.asp` (cited 2013)

[18] Jay, C., Brown, A.J., Harper, S.: Internal evaluation of the SASWAT audio browser: method, results and experimental materials, The University of Manchester (2010)

[19] Joseph Scheuhammer, M.C.: WAI-ARIA 1.0 Authoring Practices (2013), `http://www.w3.org/TR/wai-aria-practices/` (cited 2014)

[20] Linaje, M., Lozano-Tello, A., Perez-Toledano, M.A., Preciado, J.C., Rodriguez-Echeverria, R., Sanchez-Figueroa, F.: Providing RIA user interfaces with accessibility properties. Journal of Symbolic Computation 46(2), 207–217 (2011)

[21] Lourdes, M.: Toward an Equal Opportunity Web: Applications, Standards, and Tools that Increase Accessibility. In: Paloma, M., Belen, R., Ana, I. (eds.), pp. 18–26 (2011)

[22] Mesbah, A., Bozdag, E., Deursen, A.V.: Crawling AJAX by inferring user interface state changes. In: Proceedings of the 2008 8th International Conference on Web Engineering. IEEE Computer Society (2008)

[23] MiniwattsMarketingGroup. Internet Usage Statistics: The Internet Big Picture World Internet Users and Population Stats (2013), `http://www.internetworldstats.com/stats.htm` (cited 2013)

[24] NVDA. NonVisual Desktop Access (2013), `http://www.nvda-project.org/` (cited 2013)

[25] Rayson, P., Wilson, A., Leech, G.: Grammatical word class variation within the British National Corpus sampler. Language and Computers 36(1), 295–306 (2001)

[26] SaToGo, Screen reader from Serotek (2010)

[27] SuperNova. Screen Reader from Dolphin (2013), `http://www.yourdolphin.com/productdetail.asp?id=1` (cited 2013)

[28] Telerik. Telerik Widget Library, `http://www.telerik.com` (cited 2014)

[29] Thiessen, P., Chen, C.: Ajax live regions: chat as a case example. In: Proceedings of the 2007 International Cross-Disciplinary Conference on Web Accessibility (W4A), pp. 7–14. ACM, Banff (2007)

[30] VoiceOver, Screen reader from Apple (2010)

[31] W3C. Important Terms (2014), `http://www.w3.org/TR/wai-aria/terms` (cited 2014)

[32] WAI-ARIA. W3C Accessible Rich Internet Applications (2013), `http://www.w3.org/TR/wai-aria` (cited 2013)

[33] Window-Eyes, Screen Reader GW Micro (2010)

(De-)Composing Web Augmenters

Sergio Firmenich[1], Irene Garrigós[2], and Manuel Wimmer[3]

[1] LIFIA, Universidad Nacional de La Plata and CONICET Argentina
sergio.firmenich@lifia.info.unlp.edu.ar
[2] WaKe Research, University of Alicante, Spain
igarrigos@dlsi.ua.es
[3] Business Informatics Group, Vienna University of Technology, Austria
wimmer@big.tuwien.ac.at

Abstract. Immersed in social and mobile Web, users are expecting personalized browsing experiences, based on their needs, goals, and preferences. This may be complex since the users' Web navigations usually imply several (related) Web applications. A very popular technique to tackle this challenge is Web augmentation. Previously, we presented an approach to orchestrate user tasks over multiple websites, creating so-called procedures. However, these procedures are not easily editable, and thus not reusable and maintainable. In this paper, we present a complementary model-based approach, which allows treating procedures as (de)composable activities for improving their maintainability and reusability. For this purpose we introduce a dedicated UML profile for Activity Diagrams (ADs) and translators from procedures to ADs as well as back-translators to execute new compositions of these procedures. By combining benefits of end-user development for creation and model-driven engineering for maintenance, our approach proposes to have the best of both worlds as is demonstrated by a case study for trip planning.

1 Introduction

The evolution of the Web is a complex and constant process. Nowadays, immersed in social and mobile Web, users are expecting a personalized browsing experience, which adapt to their needs, goals, and preferences. One of the main limitations of how to adapt the application to each user is the current use of the Web. When performing a concrete task (e.g., organizing a trip) the user normally exceeds the application's boundaries, visiting several (related) Web applications. In cases like these, the user may feel a loss of context every time she navigates from one application to another, because the new application used has no way of tracking the previous user navigation. This missing integration, and also a lack in customization, has a deep impact in the user's browsing experience.

These limitations motivated the development of mash-ups tools [21] in order to merge a set of resources that are scattered among different websites into specialized applications. One often occurring limitation is that mash-ups are used straightforward when most of the tasks users perform are volatile and do not require the creation of entirely new applications. In the same context of managing existing Web applications, another technique that has emerged is called Web augmentation [3]. Web augmentation is the activity of navigating the Web using a "layer" over the visited websites. This layer may manipulate the original UI of existing third-party websites; in this way, users perceive an augmented website instead of the original one. Generally, these augmentations are performed on the client-side, once the content is delivered from the server. Normally, users having some kind of programming skills are the ones who develop the software artifacts that perform these augmentations. Web augmentation as a technique may be applied with different aims; from simple presentation changes to task-based Web integration mechanisms.

In [6] we presented an approach based on Web augmentation to orchestrate user tasks over multiple websites. It supports flexible processes by allowing the users to combine manual and automated tasks from a repertoire of patterns of tasks performed over the Web, creating so-called *procedures*, which are persisted in XML files. Although the tools around our previous approach allow users to record their own *procedures* by-example, and subsequently edit the details; larger editions, such as replacing several tasks with other equivalent ones or building reusable chunks, is challenging. However, this may be often required, since several large tasks such as planning a trip involve several smaller ones (book flights, hotel rooms, cars, etc.) and the requirements involved change, as well as the browsed websites. If the user wants to change a larger part of the procedure, the process order may have to be changed, additional tasks have to be intermingled, or complete procedures have to be substituted or executed in series. The importance of these aspects has been studied before in the field of Web applications [13] [17]. These are also relevant issues in the context of Web Augmentation, not only because the Web changes constantly and consequently the scripts may stop working, but also because the same script could be reused in several Web pages under the same domain [13].

Therefore, one challenging aspect for those approaches that support users tasks based on Web augmentation, is the maintenance of procedures, which has associated two dimensions: *(i)* how to reuse existing augmentation units in order to support complex scenarios (i.e., how to compose them to fulfill a larger goal), *(ii)* how to decompose subtasks and make them reusable chunks. In order to tackle these challenges, this paper extends our previous work with a modeling language based on UML Activity Diagrams (ADs) to represent the procedure's tasks involving dedicated transformations from procedures to activities. Models allow raising the abstraction level and the separation from the applications functional specification [19], which improve the reusability and maintenance of the procedures. In this way, the maintenance of existing procedures as well as the composition of new ones based on existing building blocks is supported by graphical modeling. By having the transformations from activities to procedures, we are able to execute new compositions of Web augmenters. With this approach, we combine the benefits of end-user development for creating procedures based on Web augmentation and model-driven engineering for maintaining Web augmenters to have the best of both worlds as is demonstrated by a case study for trip planning.

The remainder of this paper is as follows: Section 2 briefly summarizes our previous work on Web augmentation and introduces an example used to illustrate our approach. Section 3 elaborates on the proposed model-based approach for representing procedures. Section 4 discusses the state-of-the-art on Web augmentation, and finally, we conclude with pointers to future work in Section 5.

2 Background

Web augmentation is used for improving the user experience in several aspects. In particular, we have previously proposed an approach for supporting Web tasks by supporting users with *procedures* [6]. Procedures are programs focused on executing augmentation tasks when some user interaction is detected. These artifacts support tasks involving more than one application, and also give some mechanisms for moving information from one application to another one. In order to specify *procedures* we have previously designed a DSL based on XML that defines a procedure as a sequence of tasks. This DSL has been improved in the context of this work. The current version of the procedures metamodel is shown in Figure 1.

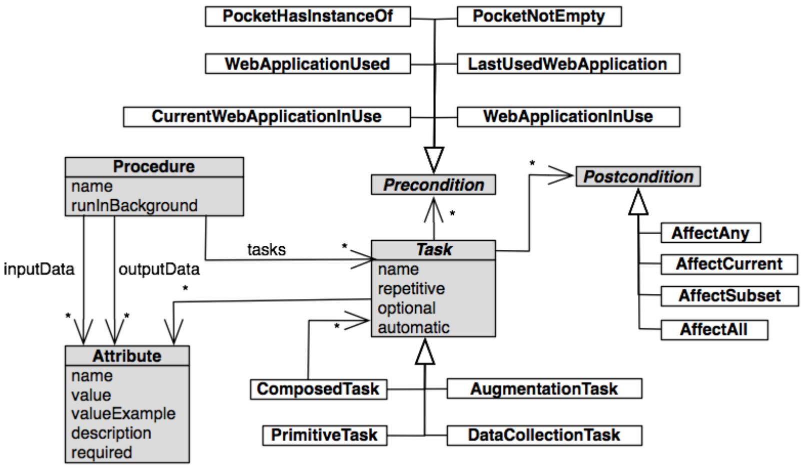


Fig. 1. The metamodel of the Web Augmentation DSL

The main concepts around the DSL are explained in the following.

- There are four types of tasks:
  - *Primitive tasks*: are based on common actions that users perform when navigating the Web (e.g., clicking an anchor).
  - *Augmentation tasks*: are tasks that allow the execution of a specific augmenter developed with our underlying framework for Web augmentation [7].
  - *DataCollection tasks*: this kind of tasks enables procedures to contemplate data collected by users. These tasks are also strongly related to *DataCollectors* and *Pocket*, two tools distributed with the framework supporting the procedures, which allow users to move information among Web applications.
  - *Composed tasks*: these tasks make possible to group other instances of tasks in order to manage them altogether. As an example, imagine the need of executing an augmenter each time that the user collects some information. In this case both tasks may be grouped in order to do repetitive the whole set.
- All tasks have three properties: *(i) repetition* property for specifying if the task may be executed more than once; *(ii) optional* property allows skipping the

execution of the task; *(iii) automatic* property is *true*, then the Web augmentation framework automatically triggers the task.

- Tasks have attributes representing information needed for the execution, e.g., if an augmenter is applied for filling in a form and it is marked as automatic, the augmenter needs to know which form fields are filled with which value.
- Tasks may have preconditions. Preconditions are used to decide if the task will be executed or not according to which information is currently available. There are two main kinds of preconditions: on the one side, *preconditions about collected data,* and on the other side, *preconditions about navigation history*.

Our approach gives support to the end-user with visual tools, deployed as Web browser plugins, for creating and executing *procedures*. Figure 2 shows the editor: a sidebar that allows users to specify tasks into the procedure while analyzing websites. The tool provides an assisted mode: users may *record* their interaction with the Web and the corresponding tasks will be added to the procedure automatically. This mode contemplates primitive tasks, augmentation tasks, and data collection tasks. Figure 3 shows how to edit a particular task. It allows users to specify the name, pre and post-conditions as well as values for both properties and attributes. If some sensitive information is saved when recording the interaction, users may remove it by editing the corresponding task.

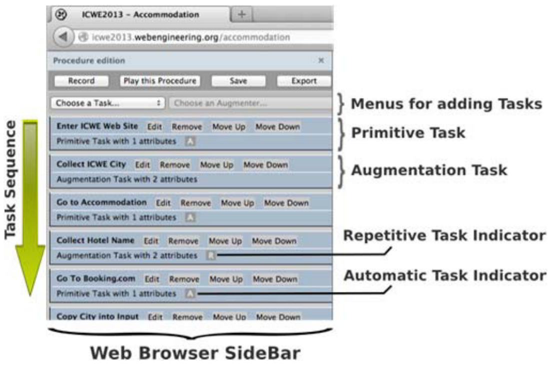
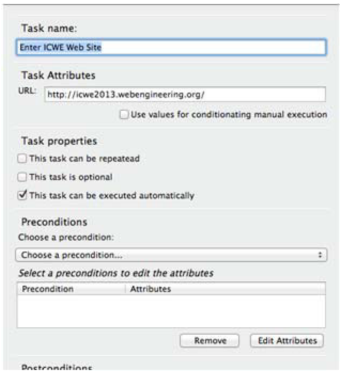

ICWE2013 - Accommodation
icwe2013.webengineering.org/accommodation
Procedure edition
Record
Play this Procedure
Save
Export
Choose a Task...
Choose an Augmenter...
Menus for adding Tasks
Enter ICWE Web Site Edit Remove Move Up Move Down
Primitive Task with 1 attributes
Primitive Task
Collect ICWE City Edit Remove Move Up Move Down
Augmentation Task with 2 attributes
Augmentation Task
Go to Accommodation Edit Remove Move Up Move Down
Primitive Task with 1 attributes
Collect Hotel Name Edit Remove Move Up Move Down
Augmentation Task with 2 attributes
Repetitive Task Indicator
Go To Booking.com Edit Remove Move Up Move Down
Primitive Task with 1 attributes
Automatic Task Indicator
Copy City into Input Edit Remove Move Up Move Down
Task Sequence
Web Browser SideBar

Fig. 2. General view of the tool


Task name: Enter ICWE Web Site

Task Attributes
URL: http://icwe2013.webengineering.org/
☐ Use values for conditionating manual execution

Task properties
☐ This task can be repeatead
☐ This task is optional
☑ This task can be executed automatically

Preconditions
Choose a precondition: Choose a precondition...
Select a preconditions to edit the attributes

Precondition	Attributes

Remove | Edit Attributes

Postconditions

Fig. 3. Edition of a single task

In order to give more insights to the real use of *procedures*, consider the following example (it will be used as a running example during the rest of the paper). The example responds to the following situation:

> *"Peter is going to travel to Paris for vacation. In that context, he has to buy flights from his town to there, and also book a taxi from the airport to downtown. For accomplishing these tasks, he uses different websites, e.g., expedia.com and wecab.com. In each of these two subtasks, Peter has to enter the same information. Besides booking flights and taxi, Peter is also interested in getting touristic information about Paris and nearby areas".*

In scenarios like this, users may take advantage of using Web augmentation approaches, since these may support users on moving relevant information from one application to other while using this information for executing augmenters in the visited websites. It is important to note that not only each augmenter is configurable (even replaceable by other similar ones) but also the subtasks (*book flight* or *book taxi*) may be also reordered and replaced according with the user's interest. Also the information used for performing the tasks (both primitive and augmentation tasks) may vary in distinct executions of the same procedure. It could be achieved by using conceptual tags during data collection tasks. In this way, if different users prefer, for example, different hotel's location or airlines, the procedure can be defined for consuming information through concept names such as "Hotel Location" or "Airline" instead of concrete data.

Although this is a common scenario, the order used for each subtasks may vary for different instantiations of the same scenario, when these are more complex. It may vary even more when Web augmentation is involved, because it is desirable to allow users to vary the augmentations applied in an easy way and to compose different procedures to solve larger examples. Thus, we provide a complementary extension to the end-user based development of Web augmenters, namely a model-based maintenance approach as explained in the next section.

3 (De-)Composing Procedures – A Model-Based Perspective

In order to solve the before mentioned drawbacks, we present a model-based approach, which allows treating procedures as composable activities. For this purpose we introduce transformations from procedures to activities as well as back-transformations to be able to execute new compositions of augmenters. In order to do so, we first need to be able to represent the procedures on the model level. With this goal in mind, we propose to use UML activity diagrams (ADs).

3.1 Model-Based Representation of Procedures

Representing procedures with ADs [16], in particular following the fUML execution semantics proposed by the OMG [15], requires a systematic mapping between our DSL and ADs. Here we follow existing methodologies for deriving UML profiles from DSL metamodels [18,20]. After investigating ADs for the purpose of modeling *procedures*, we identified a high overlap, although the later are, of course, more specific as the former. The following table illustrates the identified mappings between our DSL and ADs from a Web augmentation point of view, i.e., only the AD concepts are shown that are corresponding to the DSL concepts.

In addition to the mappings, to explicitly represent the specifics of Web augmenters (cf. Table 1 – column comments), we introduce a Web Augmentation profile for ADs. By using this profile, we are able to provide information preserving the transformations between the executable procedures expressed as XML files and the corresponding ADs. This property is one of the main building blocks of our approach to allow the continuous development on the front-end side (recording and testing procedures) as well as on the model side (maintaining and composing

Table 1. Mapping of Web Augmentation concepts to UML activity diagrams

Web Augmentation Procedures	UML Activity Diagrams	Comments
Procedure	Activity Diagram	*runInBackground* attribute has no direct mapping to UML, rest has direct mapping to UML
Task	Activity	*Optional*, *automatic*, *repetitive* attributes have no direct mapping to UML, rest has direct mapping to UML
PrimitiveTask	Activity	May be mapped to *Action* metaclass, but to allow for properties, *Activity* is used as metaclass
AugmentationTask	Activity	Same comment as for PrimitiveTask
DataCollectionTask	Activity	Same comment as for PrimitiveTask
ComposedTask	Activity	Activity may contain other activities by using CallBehaviorAction
Attribute	Property	*Value* and *example* attributes have no direct mapping to UML, rest has direct mapping to UML
Precondition	Constraint (LocalPreCondition)	*Precondition* subclasses have no direct mapping to UML
Postcondition	Constraint (LocalPostCondition)	*Postcondition* subclasses have no direct mapping to UML

procedures). Figure 4 shows the introduced stereotypes (for each meta-class we introduce a stereotype) as well as the extended metaclasses of UML. Please note that we only introduce tagged values in the profile for properties of the DSL that miss corresponding properties in the base UML metaclasses. By this, we ensure to reuse as much as possible the UML language and to keep the profile concise and minimal.

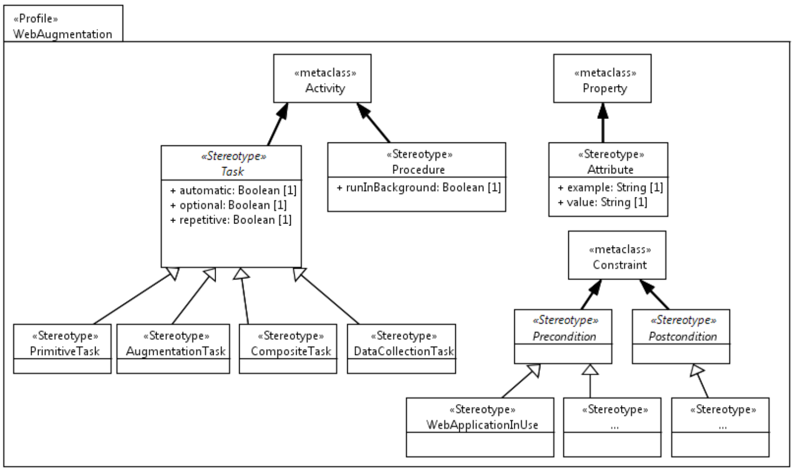

«Profile» WebAugmentation
«metaclass» Activity
«metaclass» Property
«Stereotype» Task
+ automatic: Boolean [1]
+ optional: Boolean [1]
+ repetitive: Boolean [1]
«Stereotype» Procedure
+ runInBackground: Boolean [1]
«Stereotype» Attribute
+ example: String [1]
+ value: String [1]
«metaclass» Constraint
«Stereotype» PrimitiveTask
«Stereotype» AugmentationTask
«Stereotype» CompositeTask
«Stereotype» DataCollectionTask
«Stereotype» Precondition
«Stereotype» Postcondition
«Stereotype» WebApplicationInUse
«Stereotype» ...
«Stereotype» ...

Fig. 4. Web Augmentation Profile – an Extension for UML Activity Diagrams

To summarize the syntax of the developed profile, we use the *Activity* metaclass as the main metaclass for our extension. We map the *Procedure* metaclass to the *Activity* metaclass, because ADs are UML internally represented by a root activity that contains all the elements shown within the ADs. Also the different kinds of *Task*s are mapped to the *Activity* metaclass. Because the *Activity* metaclass inherits from the

Classifier metaclass in UML, activities may contain properties. This is exactly what is required for reflecting the *Attribute* concept of *Tasks*. Besides these aspects, activities may also be nested, by using the *CallBehaviourAction* that is also able to trigger another activity from a context activity. By this, we can simulate the *CompositeTask* concept of the Web Augmentation DSL.

Moreover, we extend the metaclass *Property* with a stereotype to represent the *Attribute* metaclass of the DSL and to introduce additional attributes to allow the definition of an actual *value* and an *example value* for properties. Finally, we extend the *Constraint* metaclass of UML with specific stereotypes to reflect the specific pre- and post-condition types contemplated in our DSL.

Concerning the semantics of ADs, we consider an explicit control flow, normally a sequence of tasks, by defining *control flow links*. This is quite analogue to the sequence of tasks involved in a procedure and the information flow among these. In addition, we also exploit other control structure possibilities of ADs such as parallelization, conditions, etc. However, these constructs are not explicitly available in the current version of the Web augmentation DSL and thus, have to be compiled to a more verbose representation on the execution level. In addition to the control flow, we explicitly model the data flow, i.e., to represent the pocket and data collectors of the Web augmentation DSL, by making use of the *object flow links* supported by UML activity diagrams. By using this type of links, we are able to connect *activity parameter nodes*, i.e., parameters that are set externally before calling a certain activity as well as parameters that provide values after the execution of an activity to its environment, with so called *pins*. Activities may have input and output pins that represent input and output parameters, respectively. Pins are a powerful modeling concept in UML, e.g., by setting the multiplicity of input pins, input pins may be defined as mandatory or optional (i.e., a value is available or not for a given pin) for the execution of an activity. By linking output pins with input pins, data exchange between two activities is defined.

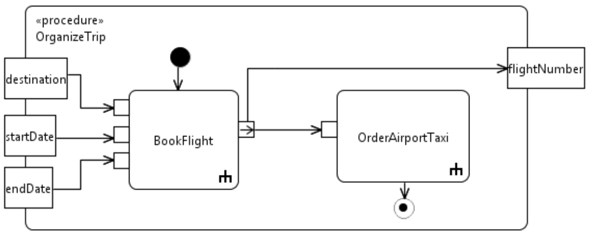


Fig. 5. Procedure OrganizeTrip in UML Activity Diagram Notation

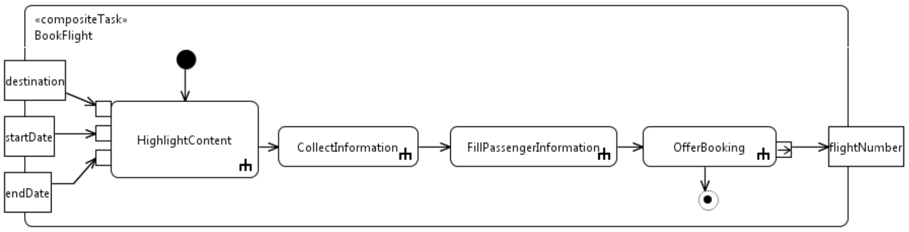


Fig. 6. Composite Task *Book Flight* as UML Activity Diagram

Consider again our main example. If we take only one of the main subtasks, such as *book flights* (a *CompositeTask* called *BookFlight*), an activity diagram with stereotype «*procedure*» is generated (cf. Figure 5) for visualizing the execution of the sequence of *tasks* as illustrated in Figure 6. In this specific case, and for reason of conciseness, we only contemplated *AugmentationTasks*, but the given activity may also include several *PrimitiveTasks* allowing the *procedure* developer to specify specific user interactions. Again we use the *CallBehaviorActions* to call the primitive and augmentation tasks.

3.2 Transformation Chain: Procedures to Activities and Back Again

In order to allow for a transparent transition from Web Augmentation (WA) DSL expressed in XML to UML activity diagrams (ADs) and back again, we implemented a bi-directional transformation chain consisting of a set of transformations as explained in the following paragraphs. More information on the implementation may be found at our project website[1].

Model Injection/Extraction Transformations. We developed an XML 2 WA DSL transformation that parses the XML-based representations and produces models conform to an Ecore-based WA DSL metamodel. In addition, we developed a WA DSL 2 XML transformation for printing models back to executable XML code. These transformations have been implemented in Groovy[2] due to its dynamic programming features and the support by the XmlSlurper and XmlMarkupBuilder APIs.

DSL/UML Integration Transformations. We developed a WA DSL 2 UML AD transformation that produces UML models from WA DSL models and applies automatically the Web augmentation profile to the UML models. In addition, we also developed the inverse transformation that takes a profiled UML model and produces a WA DSL model. These transformations have been implemented in ATL [11] due to its support for EMF models as well as UML models and the possibility to deal with profile information within the transformations.

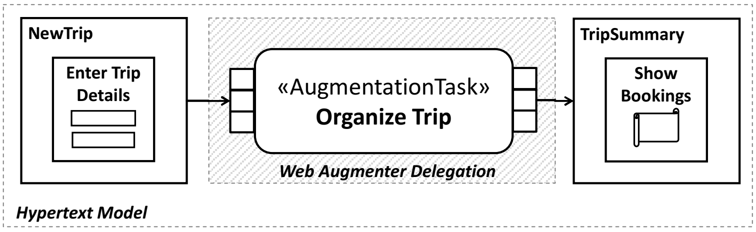


Fig. 7. Composing Web Augmentation Tasks with Hypertext Models

3.3 Composing Web Augmenter Models and Hypertext Models

One additional benefit of having Web augmenters explicitly modeled is the possibility to compose them with traditional Web design models such as supported by WebML,

[1] https://sites.google.com/site/decomposingwebaugmenters
[2] http://groovy.codehaus.org

OO-H [10], or UWE [12]. By this, Web augmentation techniques may be used by Web applications by delegating to pre-defined Web augmenters or Web augmenters may be developed for a specific Web application and integrated in the hypertext models of such applications. Consider the following example. Assume one would like to provide for a Web application that offers specific events the possibility to book a hotel room at an external website. Navigating to the external website with the specific information such as place and time may be provided by the hypertext model. This information may be passed by typical transport links transferring parameters to the Web augmenter activity (as it is done for standard hypertext nodes) and the Web augmenter activity may provide information of the booked hotel room back to the hypertext model again as parameters of a transport link. In Figure 7 we show such a composition of a hypertext model and a Web augmenter activity for the WebML language. We leave as subject for future work the creation of Web augmenter units for WebML based on the WebRatio inherent extension mechanism and the integration of the profile presented in this paper with the UWE profile for modeling hypertext models. We think this is an important line of future work to close the gap between traditional Web modeling and Web augmentation.

4 Related Work

Several approaches for supporting Web user tasks have been created, and different abstraction levels have been used. For example, CoScripter [1] proposes a DSL for supporting recurrent tasks, which may be parameterized in order to alter the data used in each step. The main idea of CoScripter is to automate some tasks by recording the user interactions (based on DOM events) and then the script may reproduce the same steps automatically. A similar approach, ChickenFoot [2], also proposes a DSL that raises the abstraction level of JavaScript programs in order to emulate user behaviour easily. However, although these approaches support slight changes in the task processes, considerable changes over these cannot be contemplated. These tools allow modifying end-user programs to vary the way that tasks are going to be performed, but usually, the *augmentation* effect is limited to a predefined subset of possibilities.

Although we share the philosophy behind these approaches, we think that further efforts should be made for making this kind of tools closer to the actual use of the Web, because users navigate the Web in a volatile way, and some tasks may be achieved in different ways (Web applications involved, data used, navigation) under different circumstances. In previous work we have presented our approach called *procedures*. Although this involves a composition of tasks where each task may be preconditioned and parameterized, the reuse of *parts* of procedures related to a particular subtask is not foreseen. All the mentioned approaches would improve taking into account some aspects from task modelling such as HAMSTERS [14], in which “abstract tasks” may be defined and the execution order may be more flexible.

The most related work in this context is [4], which proposed to model the user navigation using state machines in order to create the so-called webflows. This work defines a DSL, which allows users to specify the navigation flow as well as the data associated with each transition. One of the main differences to our work is the fact that [4] does not foresee the inclusion of third-party augmentations (i.e. developed by users), which again implies a limitation of augmentation effects. In our approach, this is contemplated by the execution of augmenters [7]. Finally, [9] define a UML profile for data mashups, but the integration with Web augmenters is not considered.

5 Conclusions and Future Work

Web augmentation is an emerging trend that allows users to improve their experiences while navigating the Web. Several approaches have been proposed to improve websites with different goals, from accessibility aspects over data integration to complex user task support, which is the focus of this work.

Although there are currently several works aiming to support specific navigation scenarios, user navigation is not always systematic as current approaches assume. In this way, one of the main challenges in this context is to support users even under volatile requirements. There are several other issues in the middle, such as how easy users may define their own artifacts for these approaches. The key is to find a good trade-off between the expressivity of the approach (what can be specified) and the usability of the tools (how it is specified). Reaching this point is challenging, and in this work, we aim to address a solution of maintaining procedures by using activity diagrams, where each activity represents a relevant subtask in a more general navigation scenario. Of course, the target users of the proposed modeling approach may no longer be end-users, but Web engineers may decompose, recompose, and maintain already existing Web augmenters and integrate these pieces in their developed hypertext models. The next steps imply defining mechanisms for including the transformations developed in this work in our Web augmentation tools and performing experiments with different kinds of users. Since our underlying Web augmentation framework allows tracking the user interaction, we plan to incorporate aspect orientation concepts [8] in order to further (de)compose procedures when cross-cutting concerns occur.

References

1. Bogart, C., Burnett, M., Cypher, A., Scaffidi, C.: End-user programming in the wild: a field study of CoScripter scripts. In: VL/HCC, pp. 39–46 (2008)
2. Bolin, M., Webber, M., Rha, P., Wilson, T.: C. Miller R.: Automation and customization of rendered web pages. In: UIST, pp. 163–172 (2005)
3. Díaz, O.: Understanding Web augmentation. In: Grossniklaus, M., Wimmer, M. (eds.) ICWE Workshops 2012. LNCS, vol. 7703, pp. 79–80. Springer, Heidelberg (2012)
4. Diaz, O., De Sosa, J., Trujillo, S.: Activity fragmentation in the Web: empowering users to support their own webflows. In: Hypertext, pp. 69–78 (2013)
5. Díaz, O., Arellano, C., Iturrioz, J.: Interfaces for Scripting: Making Greasemonkey Scripts Resilient to Website Upgrades. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 233–247. Springer, Heidelberg (2010)
6. Firmenich, S., Rossi, G., Winckler, M.: A Domain Specific Language for Orchestrating User Tasks Whilst Navigation Web Sites. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 224–232. Springer, Heidelberg (2013)

7. Firmenich, S., Winckler, M., Rossi, G., Gordillo, S.: A crowdsourced approach for concern-sensitive integration of information across the web. JWE 10(4), 289–315 (2011)
8. Garrigós, I., Wimmer, M., Mazón, J.-N.: Weaving Aspect-Orientation into Web Modeling Languages. In: Sheng, Q.Z., Kjeldskov, J. (eds.) ICWE Workshops 2013. LNCS, vol. 8295, pp. 117–132. Springer, Heidelberg (2013)
9. Gaubatz, P., Zdun, U.: UML2 Profile and Model-Driven Approach for Supporting System Integration and Adaptation of Web Data Mashups. In: Grossniklaus, M., Wimmer, M. (eds.) ICWE Workshops 2012. LNCS, vol. 7703, pp. 81–92. Springer, Heidelberg (2012)
10. Gómez, J., Cachero, C., Pastor, O.: Extending a Conceptual Modelling Approach to Web Application Design. In: Wangler, B., Bergman, L. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 79–93. Springer, Heidelberg (2000)
11. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Sci. Comput. Program. 72(1-2), 31–39 (2008)
12. Koch, N., Kraus, A., Zhang, G., Baumeister, H.: UML-Based Web Engineering - An Approach Based on Standards. In: Web Engineering, pp. 157–191 (2008)
13. Li, J., Gupta, A., Arvid, J., Borretzen, B., Conradi, R.: The empirical studies on quality benefits of reusing software components. In: COMPSAC, pp. 399–402 (2007)
14. Martinie, C., Palanque, P., Winckler, M.: Structuring and composition mechanisms to address scalability issues in task models. In: Campos, P., Graham, N., Jorge, J., Nunes, N., Palanque, P., Winckler, M. (eds.) INTERACT 2011, Part III. LNCS, vol. 6948, pp. 589–609. Springer, Heidelberg (2011)
15. Object Management Group. Unified Modeling Language (UML), Superstructure, Version 2.4.1 (2011), `http://www.omg.org/spec/UML/2.4.1`
16. Object Management Group. Semantics of a Foundational Subset for Executable UML Models (fUML), Version 1.0 (2011), `http://www.omg.org/spec/FUML/1.0`
17. Rossi, G., Schwabe, D., Lyardet, F.: Abstraction and Reuse Mechanisms in Web Application Models. In: Mayr, H.C., Liddle, S.W., Thalheim, B. (eds.) ER Workshops 2000. LNCS, vol. 1921, p. 76. Springer, Heidelberg (2000)
18. Selic, B.: A Systematic Approach to Domain-Specific Language Design Using UML. In: ISORC, pp. 2–9 (2007)
19. Van Deursen, A., Visser, E., Warmer, J.: Model-driven software evolution: A research agenda. In: Workshop on Model-Driven Software Evolution (2007)
20. Wimmer, M.: A semi-automatic approach for bridging DSMLs with UML. IJWIS 5(3), 372–404 (2009)
21. Yu, J., Benatallah, B., Casati, F., Florian, D.: Understanding mashup development. IEEE Internet Computing 12(5), 44–52 (2008)

An Exploratory Study on the Relation between User Interface Complexity and the Perceived Quality

Seyyed Ehsan Salamati Taba[1], Iman Keivanloo[2], Ying Zou[2], Joanna Ng[3], and Tinny Ng[3]

[1] School of Computing, Queen's University, Canada
[2] Department of Electrical and Computer Engineering, Queen's University, Canada
[3] IBM Toronto Lab, Markham, Ontario, Canada
taba@cs.queensu.ca, {iman.keivanloo,ying.zou}@queensu.ca, {jwng,tinny.ng}@ca.ibm.com

Abstract. The number of mobile applications has increased drastically in the past few years. Some applications are superior to the others in terms of user-perceived quality. User-perceived quality can be defined as the user's opinion of a product. For mobile applications, it can be quantified by the number of downloads and ratings. Earlier studies suggested that user interface (UI) barriers (*i.e.,* input or output challenges) can affect the user-perceived quality of mobile applications. In this paper, we explore the relation between UI complexity and user-perceived quality in Android applications. Furthermore, we strive to provide guidelines for the proper amount of UI complexity that helps an application achieve high user-perceived quality through an empirical study on 1,292 mobile applications in 8 different categories.

1 Introduction

Mobile applications are pervasive in our society and play a vital role in our daily lives. Users can perform similar tasks both on smartphones and PCs [1] such as: checking e-mails or browsing the web. Due to the limitations of smartphones (*e.g.,* small screen size, network problems and computational power) developers should be more careful in designing their applications on smartphones than PCs. Developers' negligence in the importance of UI design is one of the major reasons for users to abandon a task on smartphone and switch to PC [2].

User-perceived quality can be defined as user's opinion of a mobile application. It can be quantified by the number of downloads and ratings in mobile stores. It is important to mention that based on our definition user-perceived quality has no relation with usability in this context. The studies conducted by Karlson et al. [2] and Kane et al. [3] demonstrate that improper use of UI elements (*e.g.,* input and output) on mobile applications increases end-user frustration. For example, the excessive use of input fields in mobile applications negatively affect user-perceived quality. Although mobile applications seem to be simple and easy to develop, these studies illustrate that designing UI for mobile applications is not a trivial task.

Software metrics are widely used to derive guidelines for programmers. For example, McCabe [4] defines a complexity metric for functions, and recommends a proper implementation should hold a value below 10. Such guidelines can be exploited either during the development process for on-the-fly recommendation or during the quality assurance process. There exist several studies on the design patterns for UI development of mobile applications [5]. However, they do not provide a concrete number of appropriate UI complexity for mobile applications in order to achieve high user-perceived quality. In this paper, we focus on UI complexity and its relation with the user-perceived quality of mobile applications. Moreover, we aim to derive guidelines for UI complexity by mining the available mobile applications on Android Market. We define seven UI complexity metrics that can be calculated using static analysis. We calculate the metrics in two different granularities: i) *category*, and ii) *functionality* of 1,292 mobile applications. A category reflects the purpose of a group of mobile applications (*e.g.,* Shopping or Health) extracted from mobile stores. A functionality defines a fine-grained capability of a mobile application (*e.g.,* Payment or Sign in). We observe that there exists a relation between UI complexity and user-perceived quality of application pages (activities) belong to a similar functionality. UI complexity is dependent on the corresponding functionality. Activities with high user-perceived quality tend to be simpler in terms of UI complexity in general.

2 Background

In this section, we briefly talk about the architecture of Android applications. Android applications are written in Java programming language using Android Software Development Kit (SDK). The Android SDK compiles the code into an Android PaKage (APK) file which is an archive file with a ".apk" extension. One APK file contains all the content of an Android application.

Application components are the essential building blocks of an Android application. There are four different types of application components, including activities, services, content providers and broadcast receivers. Among those, users only interact with activities. An Android application consists of several activities. An activity is a single, focused task that the user can do. Each activity represents a single-screen user interface (UI). As a result, only one activity can be in the foreground for the users to interact with.

There are two ways to declare a UI layout for an activity: i) Declaring UI layout elements in an XML file (standard), or ii) Instantiating UI layout elements programmatically. Our premise in this work is towards the former approach since it is the recommended way by Android design guidelines [6]. Applications using the latter way are excluded from our study since our analysis and data gathering approach cannot handle them.

Every Android application has an AndroidManifest.xml (manifest) file in root directory. It contains meta-data information of an application (*e.g.,* the path to the source code of activities, permissions).

3 Study Design

3.1 Data Collection

In Android Market, there are 34 different kinds of categories from which we analyze 8 different categories. The 8 different categories are: Shopping, Health, Transportation, News, Weather, Travel, Finance and Social. Table 1 shows descriptive statistics for different categories. In total, we study 1,292 free android applications crawled in the first quarter of 2013.

Table 1. Summary of the characteristics of different categories

Category	# Applications	# Activities	# Inputs	# Outputs	# Elements
Shopping	193	2,822	12,529	25,058	68,468
Health	286	4,129	23,232	40,330	108,366
Transportation	128	1,078	5,603	7,718	22,991
News	114	1,302	4,725	7,407	23,507
Weather	244	1,608	6,713	38,659	84,739
Travel	106	1,711	7,164	15,210	38,285
Finance	103	1,167	5,989	12,899	33,818
Social	118	1,107	4,948	7,646	24,091

Extracting User-Perceived Quality. In Android Market, users can rate applications from 1 to 5 (i.e., Low to High), and write comments. The rating reflects the user-perceived quality of applications.However, Ruiz et al. [7] have shown that the rating of an application reported by Android Market is not solely a reliable quality measure. They found that 86% of the five-star applications throughout the Android Market in 2011 are applications with very few raters (less than 10 raters). Moreover, Harman et al. [8] show that the ratings have a high correlation with the download counts which is a key measure of the success for mobile applications. To overcome these challenges, we measure user-perceived quality by considering both rating and popularity factors (*i.e.,* the number of downloads and raters) using Equation (1):

$$UPQ(A) = (\frac{1}{n} * \sum_{j=1}^{n} log(Q_j)) * Rating(A). \tag{1}$$

Where $UPQ(A)$ is the measured user-perceived quality for an application; A refers to an application; n is the total number of quality attributes (i.e., the number of downloads and raters) extracted from Android Market for A. Q_j shows a quality attribute. To normalize the value of quality attributes, we used log transform. Rating(A) is the rating score extracted for A from the Android Market.

3.2 Data Processing

Extracting APK Files. To extract the content and the needed information from APKs, we use apktool [9], a tool for reverse engineering closed, binary

Android applications. It decodes APK files almost to the original form and structure. It provides the source code of the application in an intermediate "Smali" format [10] which is an assembler for the dex format used in Android Java virtual machine implementation.

Inspecting Decoded APK Files. Given an activity, there does not exist any direct mapping between its source code and its UI page. To measure UI complexity, we need to recover this linking.

Given an application, we extract the path to the source code of activities from the manifest file. To map the activities to their corresponding XML layouts, similar to Shirazi et al.'s work [11], we parse the source code of an activity (*i.e.,* Smali file) to look for a call of the *SetContentView()* method, which includes an ID to the corresponding UI XML layout file. However, this heuristic cannot map an activity to the corresponding XML layout file if the input argument to this method is the name of the UI XML layout file. To overcome this issue, we trace both IDs and names.

Calculating Metrics. We parse the XML layout files to calculate different UI metrics that is used to quantify UI complexity. We consider two sets of metrics in different granularities (*i.e.,* application and activity levels) as shown in Table 2. For the application level metrics, we compute the UI complexity metrics for each activity, and lift the metrics up to the application level by using the *average* values for ANI, ANO, ANE and *sum* for NA. We categorize the elements as inputs and outputs as shown in Table 3. We use input and output tags listed in Table 3 since such elements are frequently used in Android applications [11].

Table 2. Proposed Application and Activity Level Metrics

	Metric Names	Description
Activity Level	NI	Number of Inputs in an activity
	NO	Number of Outputs in an activity
	NE	Number of Elements in an activity
Application Level	ANI	Average Number of Inputs in an application
	ANO	Average Number of Outputs in an application
	ANE	Average Number of Elements in an application
	NA	Average Number of Activities in an application

Table 3. Input and Output Tags

	Element Names
Inputs	Button, EditText, AutoCompleteTextView, RadioGroup, RadioButton ToggleButton, DatePicker, TimePicker, ImageButton, CheckBox, Spinner
Outputs	TextView, ListView, GridView, View, ImageView, ProgressBar, GroupView

Extracting Functionalities. We extract the functionalities of each mobile application using text mining techniques. For each activity, we extract contents, strings, labels and filenames associated to the source code of activities and their corresponding UI XML layout files. We use two different heuristics to extract the texts shown to a user from an activity: i) labels assigned to each element in the UI XML layout file, and ii) strings assigned from the source code. Finally, we use LDA [12] to automatically extract the functionalities in each category.

4 Study Results

This section presents and discusses the results of our two research questions.

RQ1: Can our measurement approach quantify UI complexity?

Motivation. Measuring the complexity of a UI is not a trivial task. As the first step, we evaluate if our UI complexity metrics and our measurement approach (*i.e.,* static analysis) can be used to quantify UI complexity. We want to answer this concern by testing whether our UI complexity metrics can testify hypotheses reported by previous different studies. A user study by Kane et al. [3] has shown that user-perceived quality of some categories of mobile applications is lower than the others. For example, users are reluctant to use smartphones for shopping purposes. As a result, we aim to find out whether we can make similar observations using our metrics and approach. If we provide evidence that our measured metrics for quantifying UI complexity can correlate with the findings of previous studies, we will conjecture that our proposed metrics can be used for studies on the UI complexity of mobile applications.

Approach. For each APK file (application), we use the approach mentioned in Section 3.2 to map the source code of activities to their corresponding UI XML layout files. Next, to quantify UI complexity within each category (see Table 2), we calculate four application level UI metrics (*i.e.,* ANI, ANO, ANE and NA). Finally, based on each metric, we observe whether the UI complexity is different between categories. We test the following null hypothesis among categories:

H_0^1: *there is no difference in UI complexity of various categories.*

We perform Kruskal Wallis test [13] using the 5% confidence level (*i.e., p*-value < 0.05) among categories. This test assesses whether two or more samples are originated from the same distribution.

To testify the previous findings by Kane et al. [3], we classify our categories based on their study into two categories: i) applications that belong to the categories with high user-perceived quality, and ii) the ones that belong to categories with low user-perceived quality (*i.e.,* Shopping, Health, Travel, Finance, Social). Then, we investigate whether UI complexity is different among these two groups. We test the following null hypothesis for these two groups:

H_0^2: *there is no difference in the UI complexity of applications related to categories with high and low user-perceived quality.*

We perform a Wilcoxon rank sum test [13] to evaluate H_0^2, using the 5% level (*i.e.*, *p*-value < 0.05).

Findings. Our Approach for Quantifying UI Complexity Confirms the Findings of Previous Studies. The Kruskal Wallis test was statistically significant for each application level UI metric between different categories (Table 4)

Table 4. Kruskal-Wallis test results for application level UI metrics in different categories

Metric	p-value
ANI	0.001148
ANO	<2.2e-16
ANE	<2.2e-16
NA	4.842e-05

Table 5. Wilcoxon rank sum test results for the usage of application level UI metrics in categories with high and low user-perceived quality

Metric	p-value	ΔCliff
ANI	1.23e-11	-0.21
ANO	1.927e-10	-0.19
ANE	0.001	-0.10
NA	0.007	-0.05

meaning that there exists a significant difference in the UI complexity of various categories. Moreover, there also exists a difference between the UI complexity of applications related to categories with high and low user-perceived quality. As shown in Table 5, there exists a significant difference in UI complexity quantified by the four studied metrics that are used to quantify the applications in the categories of high and low user-perceived quality. Therefore, by quantifying UI complexity of mobile applications, we found the similar findings as the earlier user studies ([2], [3]) that UI complexity is important on user-perceived quality of mobile applications, and it varies among different categories. Therefore, our measurement approach based on static analysis can quantify UI complexity.

RQ2: Does UI complexity have an impact on the user-perceived quality of the functionalities in mobile applications?

Motivation. Mobile applications have a lot of variety even in the same category. To perform a fine-grained analysis, we cluster the activities based on their functionalities. We investigate whether there is a relation between UI complexity and the user-perceived quality among various functionalities of mobile applications. If yes, we can provide guidelines to developers of the proper number of activity level UI metrics required to have a high quality functionality.

Approach. For each application, we extract the corresponding activities and their UI XML layouts (see Section 3.2). Next, to label each activity with a fine-grained functionality, we use LDA [12] which clusters the activities (documents) based on their functionalities (*i.e.,* topics). In other words, for each activity, we extract all the strings and labels shown to the users (see Section 3.2). We apply LDA to all the activities retrieved from the existing applications in a category to extract their corresponding functionalities.

Since mobile applications perform a limited number of functionalities, the number of topics (*i.e.,* K) should be small in our research context. As we are interested in the major functionalities of applications, we empirically found that $K = 9$ is a proper number for our dataset by manual labeling and analysis of randomly selected mobile applications. We use MALLET [14] as our LDA implementation. We run the algorithm with 1000 sampling iterations, and use the parameter optimization provided by the tool to optimize α and β. In our corpus, for each category, we have n activities (extracted from the applications in the corresponding category) A = $\{a_1, ..., a_n\}$, and we name the set of our topics (*i.e.,* functionalities) F = $\{f_1, ..., f_K\}$. These functionalities are different in each category, but the number of them is the same ($K = 9$). For instance, f1 in the Shopping category is about "*Login*" and "*Sign in*" functionality. However, in the Health category, it is about "*information seeking*" functionality. LDA automatically discovers a set of functionalities (*i.e.,* F), as well as the mapping (*i.e.,* θ) between functionalities and activities. We use the notation θ_{ij} to describe the topic membership value of functionality f_i in activity a_j.

Each application (A) is consisted of several activities ($\{a_1, a_2, ..., a_n\}$), and it has a user-perceived quality calculated by Equation (1). To compute the user-perceived quality for each activity, we assign each activity the user-perceived quality obtained from the application that they belong to. All the activities from the same application acquire the same user-perceived quality. However, by applying LDA [12] each activity acquires a weight of relevance to each functionality. Therefore, the user-perceived quality for an activity can originate from two sources: i) the user-perceived quality of its corresponding application, and ii) the probability that this activity belongs to a functionality. Moreover, we use a cut-off threshold for θ (*i.e.,* 0.1) that determines if the relatedness of an activity to a functionality is important. A similar decision has been made by Chen et al. [15]. We calculate the user-perceived quality for each activity as the following:

$$AUPQ(a_j) = \theta_{ij} * UPQ(a_j), \tag{2}$$

Where $AUPQ(a_j)$ reflects the activity level user-perceived quality for activity j (a_j); θ_{ij} is the probability that activity j (a_j) is related to functionality i (f_i); UPQ(a_j) is the user-perceived quality of the application which a_j belongs to it.

For each functionality, we sort the activities based on the user-perceived quality. Then, we break the data into four equal parts, and named the ones in the highest quartile, activities with high user-perceived quality, and the ones in the lowest quartile, activities with low user-perceived quality. Finally, we investigate whether there exists any difference in the distribution of activity level UI metrics (*i.e.,* quantifiers of UI complexity in functionality level) between activities of low and high user-perceived quality. We test the following null hypothesis for each activity level UI metric in each category for each functionality:

H_0^3*: there is no difference in UI complexity between activities with low and high user-perceived quality.*

We perform a Wilcoxon rank sum test [13] to evaluate H_0^3. To control family-wise errors, we apply Bonferroni correction which adjusts the threshold p-value by dividing the number of tests (*i.e.,* 216). There exists a statistically significant difference, if p-value is less than 0.05/216=2.31e-04.

Findings. There is a significant difference between UI complexity of activities with low and high user-perceived quality. For each cell of Table 6, we report three pieces of information. Let's consider the cell related to the Shopping category for the first functionality (*i.e.,* f1) which refers to "*Login*" and "*Sign in*" functionalities, for the NI (*i.e.,* Number of Inputs) metric. In this

Table 6. Average usage of activity level UI metrics in the activities with low and high user-perceived quality for each functionality in each category. ($p<0.0002/50^{\star}$; $p<0.0002/5^{\circ}$; $p<0.0002^{+}$)

		f1	**f2**	**f3**	**f4**	**f5**	**f6**	**f7**	**f8**	**f9**
	NI	$\nearrow 2.23^{\star}$	$\nearrow 2.38^{\star}$	$\nearrow 3.92$	$\nearrow 2.83^{\star}$	$\searrow 3.89$	$\nearrow 3.22^{\star}$	$\nearrow 2.53^{\star}$	$\searrow 3.44^{\star}$	$\nearrow 2.25$
Shopping	NO	$\nearrow 4.01^{\star}$	$\nearrow 3.55$	$\nearrow 4.77^{\star}$	$\nearrow 4.92^{\star}$	$\searrow 8.55$	$\nearrow 5.40^{\star}$	$\nearrow 4.28^{\star}$	$\nearrow 6.27^{\star}$	$\nearrow 3.57$
	NE	$\nearrow 11.13^{\star}$	$\nearrow 9.84^{\star}$	$\nearrow 16.17^{\star}$	$\nearrow 13.32^{\star}$	$\searrow 22.80$	$\nearrow 15.71^{\star}$	$\nearrow 11.83^{\star}$	$\nearrow 16.90^{\star}$	$\nearrow 10.00^{\star}$
	NI	$\nearrow 2.92^{\star}$	$\nearrow 2.01^{\star}$	$\searrow 2.57^{\star}$	$\nearrow 3.25^{\star}$	$\nearrow 2.41^{\star}$	$\nearrow 2.54^{\star}$	$\searrow 2.55$	$\nearrow 3.23$	$\searrow 2.46$
Health	NO	$\searrow 4.20^{\star}$	$\searrow 3.16^{\star}$	$\nearrow 3.22^{\star}$	$\nearrow 5.24^{\star}$	$\nearrow 2.70^{\star}$	$\nearrow 3.70^{\star}$	$\nearrow 3.78^{\star}$	$\nearrow 4.66^{\star}$	$\searrow 3.11^{\star}$
	NE	$\nearrow 13.41^{\star}$	$\searrow 13.43^{\star}$	$\nearrow 10.35^{\star}$	$\nearrow 14.55^{\star}$	$\nearrow 8.32^{\star}$	$\nearrow 10.89^{\star}$	$\nearrow 10.86^{\star}$	$\nearrow 13.77^{\star}$	$\searrow 8.95^{\star}$
	NI	$\nearrow 2.63^{\star}$	$\searrow 2.36^{\star}$	$\searrow 3.25$	$\nearrow 2.50^{\star}$	$\nearrow 2.21^{\star}$	$\nearrow 2.45^{\star}$	$\nearrow 3.26^{\star}$	$\nearrow 2.13^{\star}$	$\searrow 2.47^{\star}$
News	NO	$\searrow 3.70^{\star}$	$\searrow 3.15^{\star}$	$\nearrow 3.50^{\star}$	$\searrow 3.03^{\star}$	$\nearrow 3.00^{\star}$	$\nearrow 3.91^{\star}$	$\nearrow 3.58^{\star}$	$\nearrow 2.51^{\star}$	$\nearrow 3.72^{\star}$
	NE	$\searrow 11.66^{\star}$	$\searrow 10.40^{\star}$	$\searrow 11.94^{\star}$	$\searrow 9.69^{\star}$	$\nearrow 10.06^{\star}$	$\nearrow 12.38^{\star}$	$\nearrow 12.59^{\star}$	$\nearrow 8.63^{\star}$	$\nearrow 12.75^{\star}$
	NI	$\searrow 3.49^{\star}$	$\nearrow 4.17$	$\nearrow 3.11^{\star}$	$\searrow 1.77^{\star}$	$\searrow 3.83^{\star}$	$\nearrow 4.39$	$\nearrow 3.22^{\star}$	$\nearrow 4.09^{+}$	$\nearrow 3.78^{\star}$
Transportation	NO	$\nearrow 2.81^{\star}$	$\nearrow 5.07^{+}$	$\nearrow 2.84^{\star}$	$\nearrow 3.35^{\star}$	$\searrow 3.49^{\star}$	$\searrow 4.21^{\star}$	$\nearrow 4.85^{\star}$	$\nearrow 3.53^{\star}$	$\searrow 5.25^{\circ}$
	NE	$\nearrow 10.39^{\star}$	$\nearrow 15.91^{+}$	$\nearrow 8.96^{\star}$	$\nearrow 9.44^{\star}$	$\nearrow 12.83^{\star}$	$\searrow 12.72^{\star}$	$\nearrow 13.34^{\star}$	$\nearrow 12.70^{\star}$	$\searrow 12.98^{\star}$
	NI	$\nearrow 2.08^{\star}$	$\searrow 3.33$	$\nearrow 2.35$	$\nearrow 1.23^{\star}$	$\nearrow 2.87^{\star}$	$\searrow 2.04$	$\nearrow 2.03^{\star}$	$\nearrow 3.78^{+}$	$\searrow 3.29^{\circ}$
Weather	NO	$\searrow 5.35^{\star}$	$\searrow 6.17^{\star}$	$\nearrow 5.57^{\star}$	$\searrow 11.77^{\star}$	$\searrow 5.48^{\star}$	$\nearrow 4.23^{\star}$	$\nearrow 1.82^{\star}$	$\nearrow 6.38^{\star}$	$\nearrow 3.79^{\star}$
	NE	$\searrow 10.92^{\star}$	$\searrow 16.61^{\star}$	$\nearrow 14.03^{\star}$	$\searrow 30.97^{\star}$	$\nearrow 14.15^{\star}$	$\nearrow 8.99^{\star}$	$\nearrow 5.92^{\star}$	$\nearrow 14.65^{\star}$	$\nearrow 11.51^{\star}$
	NI	$\nearrow 3.36^{\star}$	$\searrow 3.81$	$\nearrow 2.36^{\star}$	$\searrow 3.52^{+}$	$\nearrow 3.51^{\star}$	$\searrow 2.87^{\star}$	$\nearrow 3.64^{\star}$	$\nearrow 3.03^{\star}$	$\nearrow 2.41^{\star}$
Travel	NO	$\nearrow 4.22^{\star}$	$\nearrow 5.64^{\star}$	$\nearrow 2.61^{\star}$	$\searrow 5.66^{\star}$	$\nearrow 4.77^{\star}$	$\searrow 4.03^{\star}$	$\nearrow 4.26^{\star}$	$\searrow 3.57^{\star}$	$\searrow 3.21^{\star}$
	NE	$\nearrow 12.94^{\star}$	$\nearrow 16.62^{\star}$	$\nearrow 7.78^{\star}$	$\searrow 16.19^{\star}$	$\nearrow 14.52^{\star}$	$\searrow 12.38^{\star}$	$\nearrow 11.94^{\star}$	$\searrow 12.68^{\star}$	$\nearrow 10.46^{\star}$
	NI	$\nearrow 3.38^{\star}$	$\searrow 2.81^{\star}$	$\nearrow 3.97^{\star}$	$\searrow 2.81^{\star}$	$\nearrow 2.40^{\star}$	$\nearrow 2.37^{\star}$	$\searrow 4.14$	$\nearrow 4.02^{\star}$	$\searrow 5.29$
Finance	NO	$\nearrow 6.42^{\star}$	$\searrow 4.59^{\star}$	$\searrow 7.85^{\star}$	$\nearrow 6.30^{\star}$	$\nearrow 4.00^{\star}$	$\nearrow 4.01^{\star}$	$\nearrow 7.22^{\star}$	$\nearrow 5.98^{\star}$	$\searrow 9.41^{\circ}$
	NE	$\nearrow 15.90^{\star}$	$\searrow 11.60^{\star}$	$\searrow 21.00^{\star}$	$\nearrow 18.16^{\star}$	$\searrow 11.24^{\star}$	$\nearrow 10.58^{\star}$	$\nearrow 18.24^{\star}$	$\nearrow 16.85^{\star}$	$\searrow 24.77$
	NI	$\nearrow 2.77^{\star}$	$\searrow 3.17^{\star}$	$\nearrow 2.48^{\star}$	$\searrow 2.04^{\star}$	$\nearrow 2.96^{+}$	$\nearrow 3.02^{\star}$	$\nearrow 3.12^{\star}$	$\nearrow 2.06^{\star}$	$\searrow 4.07$
Social	NO	$\nearrow 4.57^{\star}$	$\searrow 3.81^{\star}$	$\nearrow 3.86^{\star}$	$\nearrow 2.56^{\star}$	$\nearrow 3.50^{\star}$	$\nearrow 3.86^{\star}$	$\nearrow 5.38^{\star}$	$\nearrow 2.55^{\star}$	$\searrow 6.16$
	NE	$\nearrow 14.45^{\star}$	$\searrow 14.10^{\star}$	$\nearrow 13.43^{\star}$	$\searrow 9.39^{\star}$	$\nearrow 12.41^{\star}$	$\nearrow 14.36^{\star}$	$\nearrow 16.38^{\star}$	$\nearrow 9.35^{\star}$	$\searrow 19.34$

cell, first, there is a "$\nearrow$" or "$\searrow$" sign which implies whether the difference for the corresponding metric (*i.e.,* NI) between activities with low and high user-perceived quality is positive or negative. In this example, it is positive ("$\nearrow$") which means that activities related to this functionality (f1) in the Shopping category with low user-perceived quality have more complexity for NI than the ones with high user-perceived quality. Moreover, we report the average usage of the corresponding metric (NI) for the activities with high user-perceived quality which is 2.23 in this example. Such average values can be used to derive software development guidelines (*e.g.,* McCabe [4]). Here, it implies the average number of the corresponding activity level UI metric for high quality activities. Finally, we report whether the difference in the usage of the corresponding metric (*i.e.,* NI) is statistically significant between low quality activities and high quality ones. In this example, the difference is statistically significant ($\star$).

As it can be seen from Table 6, we can reject H_0^3, and conclude that there exists a significant difference in UI complexity between activities with low and high user-perceived quality. Furthermore, we observe that UI complexity is dependent on the corresponding functionality and category. For some functionalities higher UI complexity can result in a better user-perceived quality. However, in some cases this relation is quite different. In most cases this difference is a positive number ("$\nearrow$") meaning that low quality activities tend to use more activity level UI metrics than the high quality ones. In other words, simpler activities in terms of our used activity level UI metrics may results in a better perceived quality by the users. Our guidelines can be exploited by developers to use the proper UI complexity required to have functionalities with high user-perceived quality.

5 Threats to Validity

We now discuss the threats to validity of our study following common guidelines for empirical studies [16].

Construct validity threats concern the relation between theory and observation. They are mainly due to measurement errors. Szydlowski et al. discuss the challenges for dynamic analysis of iOS applications [17]. They mention that these challenges are user interface driven. Due to such challenges, we were not able to use dynamic analysis for mobile application UI reverse engineering for a large scale study. In this study, our premise is based on the UI elements declared in XML files since it is the recommended approach by Android guidelines [6].

Threats to internal validity concern our selection of subject systems, tools, and analysis method. The accuracy of apktool impacts our results since the extracted activity and XML files are provided by this tool. Moreover, the choice of the optimal number of topics in LDA is a difficult task. However, through a manual analysis approach, we found that in all categories there exist at least 9 common functionalities.

Conclusion validity threats concern the relation between the treatment and the outcome. We paid attention not to violate assumptions of the constructed statistical models; in particular we used non-parametric tests that do not require any assumption on the underlying data distribution.

Reliability validity threats concern the possibility of replicating this study. Every result obtained through empirical studies is threatened by potential bias from data sets [18]. To mitigate these threats we tested our hypotheses over 1,292 mobile applications in 8 different categories. We chose these categories since they contain both categories with high and low user-perceived quality, and they are from different domains. Also, we attempt to provide all the necessary details to replicate our study.

Threats to external validity concern the possibility to generalize our results. We try to study several mobile applications (1,292) from different categories. Our study analyzes free (as in "no cost") mobile applications in 8 different categories of the Android Market. To find out if our results apply to other mobile stores and mobile platforms, we need to perform additional studies on those environments.

6 Conclusion

In this paper, we provided empirical evidence that UI complexity has an impact on user-perceived quality of Android applications. To quantify UI complexity, we proposed various UI metrics. Then, we performed a detailed case study using 1,292 free Android applications distributed in 8 categories, to investigate the impact of UI complexity on user-perceived quality of mobile applications. The highlights of our analysis include: i) We can quantify UI complexity based on our measurement approach (**RQ1**) and ii) There is a significant difference between UI complexity of activities with low and high user-perceived quality. Activities with high user-perceived quality tend to use less activity level UI metrics (*i.e.,* simpler) than activities with low user-perceived quality. Moreover, we derive guidelines for the proper amount of UI complexity required to have functionalities with high user-perceived quality (**RQ2**).

In future work, we plan to replicate this study on more categories existing on Android Market. Moreover, we should investigate whether our findings are consistent among other platforms (iOS and BlackBerry).

References

1. Karlson, A.K., Meyers, B.R., Jacobs, A., Johns, P., Kane, S.K.: Working overtime: Patterns of smartphone and pc usage in the day of an information worker. In: PerCom (2009)
2. Karlson, A.K., Iqbal, S.T., Meyers, B., Ramos, G., Lee, K., Tang, J.C.: Mobile taskflow in context: A screenshot study of smartphone usage. In: SIGCHI (2010)
3. Kane, S.K., Karlson, A.K., Meyers, B.R., Johns, P., Jacobs, A., Smith, G.: Exploring cross-device web use on pcs and mobile devices. In: Gross, T., Gulliksen, J., Kotzé, P., Oestreicher, L., Palanque, P., Prates, R.O., Winckler, M. (eds.) INTERACT 2009. LNCS, vol. 5726, pp. 722–735. Springer, Heidelberg (2009)
4. McCabe, T.: A complexity measure. IEEE Transactions on Software Engineering SE-2(4), 308–320 (1976)
5. Nilsson, E.G.: Design patterns for user interface for mobile applications. Advances in Engineering Software 40(12), 1318–1328 (2009)
6. Android guidelines (April 2014), `http://developer.android.com/guide/developing/building/index.html`
7. Mojica Ruiz, I.J.: Large-scale empirical studies of mobile apps. Master's thesis, Queen's University (2013)
8. Harman, M., Jia, Y., Zhang, Y.: App store mining and analysis: Msr for app stores. In: MSR (2012)
9. apktool, `http://code.google.com/p/android-apktool/`
10. smali, `http://code.google.com/p/smali/`
11. Sahami Shirazi, A., Henze, N., Schmidt, A., Goldberg, R., Schmidt, B., Schmauder, H.: Insights into layout patterns of mobile user interfaces by an automatic analysis of android apps. In: SIGCHI (2013)
12. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. The J. of Machine Learning Research 3, 993–1022 (2003)
13. Sheskin, D.J.: Handbook of parametric and nonparametric statistical procedures. CRC Press (2003)
14. McCallum, A.K.: Mallet: A machine learning for language toolkit (2002), `http://mallet.cs.umass.edu`
15. Chen, T.-H., Thomas, S.W., Nagappan, M., Hassan, A.E.: Explaining software defects using topic models. In: MSR (2012)
16. Yin, R.K.: Case study research: Design and methods, vol. 5. Sage (2009)
17. Szydlowski, M., Egele, M., Kruegel, C., Vigna, G.: Challenges for dynamic analysis of ios applications. In: iNetSec (2012)
18. Menzies, T., Greenwald, J., Frank, A.: Data mining static code attributes to learn defect predictors. IEEE Transactions on Software Engineering 33(1), 2–13 (2007)

Beyond Responsive Design: Adaptation to Touch and Multitouch

Michael Nebeling and Moira C. Norrie

Department of Computer Science, ETH Zurich
CH-8092 Zurich, Switzerland
{nebeling,norrie}@inf.ethz.ch

Abstract. The new generation of touch devices are often used for web browsing, but the majority of web interfaces are still not adapted for touch and multi-touch interaction. Using an example of an existing web site, we experiment with different adaptations for touch and multi-touch. The goal is to inform the design of a new class of web interfaces that could leverage gesture-based interaction to better support application-specific tasks. We also discuss how current responsive design techniques would need to be extended to cater for the proposed adaptations.

Keywords: responsive web design, adaptation to touch and multi-touch.

1 Introduction

Given the proliferation of touch devices, web applications in particular are increasingly accessed using input modalities other than mouse and keyboard. However, to date many web sites still do not provide an interface that is optimised for touch input, and multi-touch interaction is generally still limited to gestures for scrolling and zooming content as interpreted by web browsers [1,2]. This paper aims to show how web sites could be adapted and instead provide carefully designed multi-touch features that are tailored to the web interface and therefore of potential benefit when carrying out application-specific tasks.

We investigate the adaptation to touch and multi-touch as a two-layered web design problem with a new set of technical and design challenges beyond responsive design [3]. In the original proposal[1], responsive web design was conceived as a way of developing flexible web page layouts that can dynamically adapt to the viewing environment by building on fluid, proportional grids and flexible images. Similar to some of the techniques used to support the study presented in this paper, this is mainly based on using CSS3 media queries for defining breakpoints for different viewing conditions and switching styles to adapt the layout. While responsive design is nowadays used as a broader term to describe techniques for delivering optimised content across a wide range of devices, the focus is still on dealing with different screen sizes, rather than adapting to different input modalities such as touch, which is the focus of this paper. While both

[1] `http://alistapart.com/article/responsive-web-design`

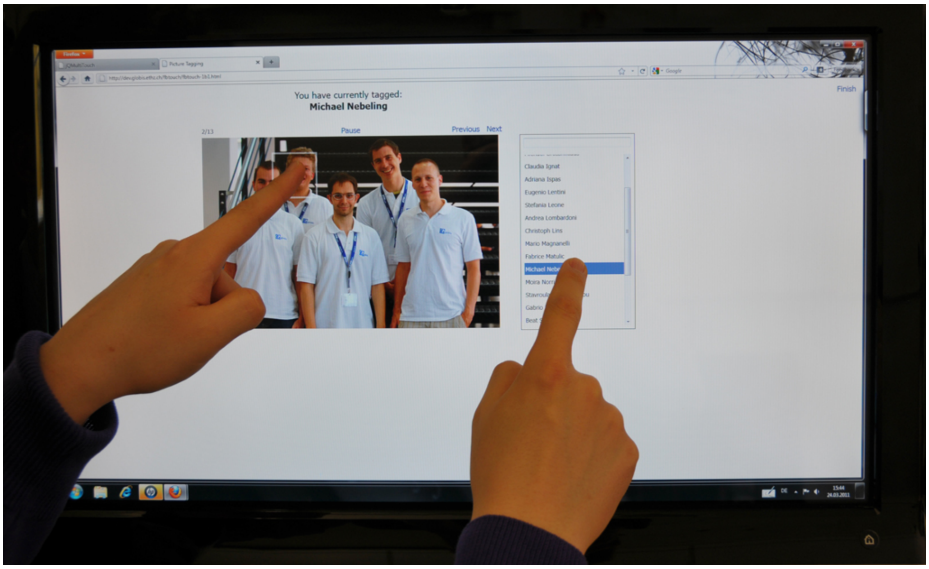


Fig. 1. One of the multi-touch versions we have designed and evaluated for a simple picture tagging application similar to Facebook, here using a two-point interaction

issues come together when designing for small, mobile touch devices, we want to separate the concerns by focusing on the adaptation to touch and multi-touch on devices where screen size is not the primary issue.

As one example, we examine alternative designs with a number of touch enhancements and different multi-touch features for *FBTouch*, a simple picture tagging application that we designed and gradually adapted based on the one provided by Facebook (Fig. 1). The goal of our work is two-fold: First, we want to identify the key issues related to touch specifically in a web context and consider various aspects of web design and the adaptation of interfaces for touch input. Second, we want to demonstrate how a new class of web interfaces with active support for multi-touch could be designed to improve task performance and the overall user experience on touch devices.

We begin by discussing related work and the background to our study in the next section. We then present the different designs created for the picture tagging application and the results of an initial user experiment. The paper discusses the implications for designing the required forms of adaptation and technical considerations for extending current responsive design techniques.

2 Background

Over the years, many different web design guidelines and best practices have been developed by practitioners and experts as well as in research. For example, several metrics to quantify usability factors such as the total word count in a page, the number of links and media as well as the spectrum of colours and font styles have been proposed [4]. More directed guidelines such as WCAG, the Web Content Accessibility Guidelines by W3C, consist of a set of recommendations on making content accessible, primarily for users with visual impairments. In our previous study, we developed a new set of metrics that address spatial factors and the distribution of content depending on the viewing condition [5]. However, the specifics of the new generation of touch devices have been out of scope of this and other studies. As a matter of fact, best practices are currently mostly driven by vendors, e.g. Apple's iOS Human Interface Guidelines.

As for the adaptation of web interfaces to different devices, previous research has focused on design issues with respect to small screens, e.g. [6], and fully automated methods for retargeting existing web interfaces to mobile phones [7]. For more comprehensive techniques, research has mainly looked at different models of user interface abstraction and model-driven approaches for generating interfaces adapted to different user, platform and environment contexts [8]. Although the authoring of adaptive and multi-modal user interfaces has been the subject of extensive research, e.g. [9], the concrete design and layout requirements have received relatively little attention. Specifically for touch, the key adaptation techniques have not been established, and studies so far have been limited to touch without considering multi-touch [2]. Despite the increasing availability of multi-touch web development frameworks such as jQMultiTouch [1], we are still far from an advanced use of multi-touch gestures in web interfaces. Rather, users currently employ simple pinch and pan gestures primarily for scrolling web pages and navigating between them, or as a workaround, and then for dealing with low-level issues such as precise selection of hyperlinks and other forms of active web content that are often inappropriately sized and placed for touch.

3 FBTouch

The initial design of our *FBTouch* example application is based on the original Facebook design. Rather than creating a new touch interface specifically designed for our study, we wanted to experiment with an existing interface with which many users are familiar, and study the effects of our adaptations. We chose picture tagging primarily because it is a common interaction technique for many Web 2.0 sites and current support in web interfaces is rather limited. As challenging and beneficial as it is for multi-touch interaction, it is also an example of a real-world task likely to be performed by users.

Common to all our FBTouch prototypes shown in Fig. 2 is the fact that users have to select from a list of names to tag people in pictures. Apart from the fact that this is the case in Facebook, we intentionally designed it in this way as it requires precise selection from multiple options which is difficult if the size and spacing of elements is too narrow for touch input. Also similar to Facebook, text search functions are provided in all interfaces to narrow down the list of names to only those that match the input. However, these are not necessarily needed because of the limited amount of scrolling required to view all name tags. The intention was not to give too much of an advantage to the mouse and keyboard

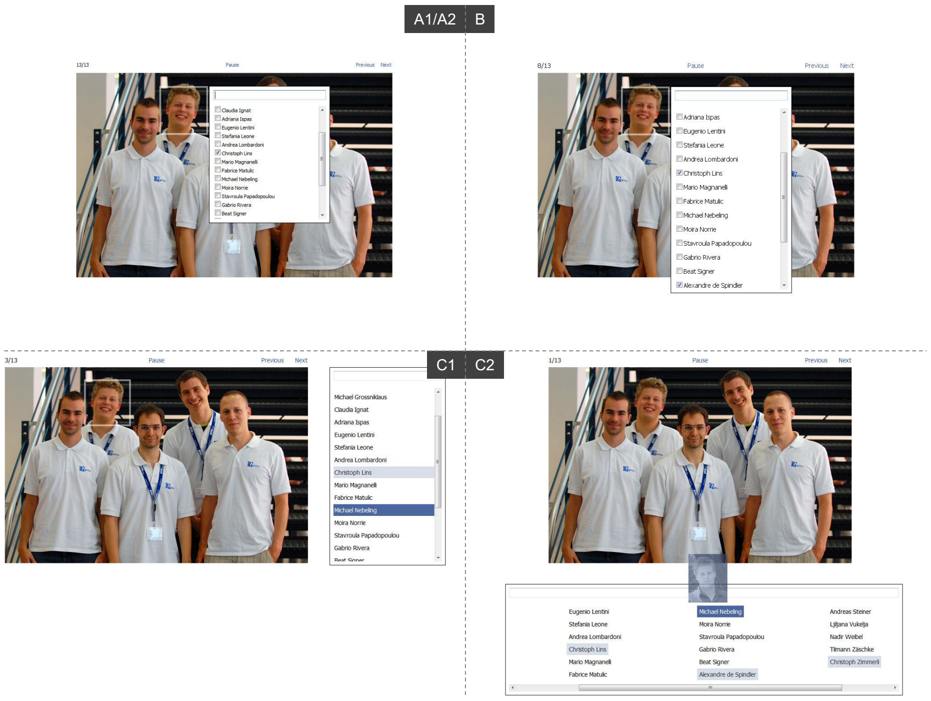


Fig. 2. Different interfaces designed and compared as part of the experiment

interface since text input on a touch screen requires users to switch between the browser and the on-screen keyboard and text input via a physical keyboard is usually faster. The alignment of the two links, "Previous" and "Next", next to each other above the picture is also similar to Facebook. At the same time, it is interesting for the purpose of our study since this alignment requires precise selection from a horizontal set of options where padding and margin are often less generous in web interfaces.

In the following, we will start by presenting the features of the standard interface and then show how it was gradually adapted, focussing mostly on what was changed between versions in terms of the tagging interaction.

The standard interface can be operated using mouse and keyboard or touch input. The tagging interaction is started by pointing with the mouse and clicking on people in the picture, or directly by touching the picture. This will then open a pop-up window next to where the input occurred, showing the search field on top and a list of names with a vertical scrollbar below. Clicking or touching the check boxes or names in the list will tag the selected person, as indicated by the checkmarks, or remove the tag if the name was already selected in a previous tagging interaction. Tagging can be cancelled by pressing ESC as well as by clicking or touching somewhere outside the pop-up window. The pop-up will be hidden after a person was successfully tagged or if the tagging interaction is cancelled. Pressing the "Previous" and "Next" links will navigate between pictures. Alternatively, the left and right arrow keys can be used to navigate to the previous or next picture. We will refer to this interface as A1 for mouse and keyboard input or A2 for touch.

Interface B uses the same tagging interaction, but in preparation for touch input, scales text slightly larger and increases the padding and spacing for active content areas such as links. In particular, the size of the "Next" link was adjusted to match the touch area of the "Previous" link. While this may not be as important for mouse input, it was considered because the touch area for the dominant action "Next" in the standard interface is significantly smaller, simply because the text link consists of only four letters. Also, the design now uses increased line height and spacing in the list of names, which required further adjustments of the list's height to have the same number of visible names. To prevent undesired default behaviour in browsers when using touch, the interface was further modified to disable text selection within touch-sensitive areas and prevent users from accidentally dragging the picture when touching it. Except for these small enhancements for touch input, the interface does not make use of any multi-touch features yet.

Interface C1 is based on the touch enhancements from the previous interface B, but in addition introduces a simple set of basic multi-touch gestures. These include swipe right or left to navigate to the previous or next picture, spread to overlay a larger version of the picture in higher quality and pinch to hide the overlay again. The tagging interaction itself then requires two-point interaction with one finger touching the picture and the other a name in the list. The order in which the touches occur determines the meaning of the interaction. Tapping the picture at different positions while touching the same name only changes the position of the tagging box. Tapping other names while touching the picture overwrites the current tag with the currently selected name. Tagged names are marked with a background colour slightly lighter than the highlight colour, and untagging can be performed by simply tapping a marked name. Interestingly, Windows 7 used on the TouchSmart with which the interfaces were developed and tested, did not allow for simultaneous touches on the picture while interacting with Windows standard controls such as the list of tags. We therefore enhanced the scrolling mechanisms of the list control to support scrolling when users touch the picture at the same time and prevent accidental tagging/untagging when scrolling occurred prior to the interaction.

The last interface C2 uses an alternative design of version C1 so that tagging now requires dragging a name from the list and dropping it on a person shown in the picture; dropping the tag outside the picture will cancel the operation. The interface supports the simple gesture set introduced in the previous version and additionally allows for performing multiple such drag-n-drop operations at a time. Not to remove names from the list via drag-n-drop, we implemented a way for touch events to be delegated to other elements that was previously not available in browsers. We used this method to drag a thumbnail of the person's photo as an intermediate representation of the original touch target. We also developed our own event capture technique so that simultaneous dragging of two or more photo tags using multi-finger or hand interaction can be supported. This new interaction is further enabled by the horizontal layout of the list now placed below the picture. Finally, the scrolling mechanisms of the previous version were adapted for horizontal scrolling not to interfere with active dragging operations, which required special event handling mechanisms.

We are aware of the fact that the design space for adaptations to support touch and gesture-based interaction is very large and that the different touch interfaces we created represent only two possible adaptations of the Facebook interface for touch devices. Nevertheless, the intention was to make only minor design modifications and then test their effects on users.

The first multi-touch tagging interaction using a two-point concept was designed as an alternative to the pop-up window from the standard versions of the interface as well as a simple way of employing two-touch to interact with two interface controls at a time. The intended meaning of this interaction was a natural mapping of pointing at somebody while calling out the name, or linking elements by holding them at the same time. However, this kind of layout that assigns rather fixed roles to hands also requires an alternative design for left-handed users who may prefer to use their left hand for the selection task.

The design modifications for the second multi-touch interface were made to allow users to use two hands independently and also to see whether users would employ multi-drag to improve their performance in the picture tagging task. The fact that this interface shows slightly more options in the list of names has two reasons. First, pilot testing showed that users are not as comfortable with the horizontal scroll layout and, second, it is also a countermeasure to make up for the fact that usually one hand hides a significant portion of the list when the tagging interaction is performed.

The swipe gestures available in both multi-touch interfaces were added to provide users with basic gestural support also known from other multi-touch applications. However, we intentionally kept the “Previous” and “Next” links in the interfaces primarily to assess the gestural support as an optional feature and make switching between them easier for users since the main navigation controls are present in every interface. Also we were interested to see how often users would make use of the gestures or the basic controls. Finally, the pinch/spread gestures for zooming only the picture rather than the entire interface were added to see whether users would appreciate an adapted zoom for the task.

The device used both for the active development as well as for user testing was an HP TouchSmart 600-1200 with 58.4 cm (23”) screen diagonal and 16:9 wide-format screen at full HD resolution (1920x1080 pixels). The multi-touch features were built on top of Firefox which has included support for Windows 7 touch events since Version 4. The FBTouch prototypes were implemented using jQMultiTouch [1] and are published on the project web site[2].

[2] `http://dev.globis.ethz.ch/fbtouch`

4 User Experiment

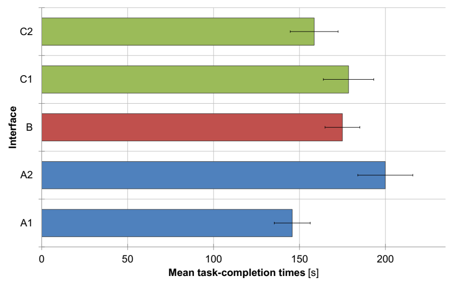
To assess the user performance and experience of the different interfaces shown in Fig. 2, we carried out a small lab study with 13 participants on the TouchSmart all-in-one computer also used for designing the interfaces. The majority of participants were between 25 and 35 years of age and right-handed. Participants' overall background using touch input was generally high and more than half of them (8 people) stated that they use touch devices several times a week to every day. The experiment consisted of performing the same simple picture tagging task with every interface. Rather than actually connecting to Facebook and tagging personal pictures and friends, we used a pre-defined set of pictures and names to make for equal conditions for all participants. The order was randomised and counterbalanced so that participants were not necessarily provided with gradually more features. They were therefore given a short time to get familiar with each interface before the actual experiment started.

The results of the user study reveal a number of interesting aspects concerning task performance and rated user experience for the different interfaces and input modalities that were tested. In general, one can see that participants were most efficient using mouse and keyboard, but the touch enhancements and multi-touch prototypes were well received by participants and helped them to achieve a better performance on the touch screen. Also, the relatively high user experience of the mouse and keyboard interface was only matched by the multi-touch interfaces. In the following, we provide a brief analysis.

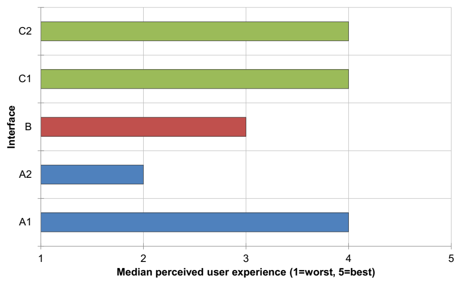
As for task performance, Figure 3a shows the average times required by participants. A one-way repeated measures ANOVA found a significant effect of time required to complete the task using the different interfaces. The mean task completion times were best for interface A1 using mouse and keyboard. On the other hand, the same interface using touch provided the worst performance as participants took on average almost one minute longer to complete the task. As a result, the differences between interfaces A1 and A2 were significant.

Quite promising when comparing interface A2 to the adapted interfaces for touch is the fact that the relatively simple touch enhancements in interface B already contributed to almost 15% faster task completion times. Also, the second best mean time overall was only then achieved as participants used the drag-n-drop multi-touch features of interface C2, which was significantly faster compared to interface A2. The mean task completion time using interface C1 was close to B. Overall, interface C2 seemed the best version for touch and multi-touch in terms of the time required to complete the picture tagging task, but the differences to B and C1 were not significant.

Moreover, interface A1 showed significant differences compared to interfaces A2, B and C1. Even though participants were on average still 9% slower with interface C2 than using mouse and keyboard, there was no significant difference between interfaces A1 and C2. This high suitability of interface C2 for touch input is also underlined by the fact that participants completed the task almost 26% faster than with interface B on the touch screen. We can therefore say that only complementing the simple touch enhancements of interface B with the



(a) Mean task-completion times



(b) Median user experience

Fig. 3. Compared to the mouse and keyboard interface A1, only interfaces C1 and C2 with task-specific adaptations for multi-touch produced similar task-completion times and user experience—the basic touch enhancements in B were not sufficient

multi-touch features of interface C2 helped participants to achieve accuracies that come a lot closer to the mouse and keyboard interface A1.

In terms of user feedback, Figure 3b illustrates that interfaces A1, C1 and C2 seemed to provide the best user experience for participants. The standard interface A2 executed on the touch display was rated by far the lowest. While interface B performed slightly better than interface C1, the user experience of the latter was on average rated considerably higher. This supports the general rule that faster execution times do not necessarily reflect in higher user experience. As a matter of fact, for nearly half of participants, interface A1 with mouse and keyboard was the fastest (6 times). Taking the touch interfaces only, interface C2 was first (5 times), closely followed by interface B (4 times). However, when looking at the best interfaces in terms of the user experience as rated by participants, we are presented with a slightly different picture where interface C1 was selected six times and interface C2 only four times.

In general, most participants felt very comfortable and fairly efficient with the touch adaptations specifically designed for the Facebook interface. When using the multi-touch interfaces C1 and C2, all participants but one favoured the swipe gestures rather than clicking the "Previous" and "Next" links for navigating between pictures. The majority of participants (10 people) also found the tagging interactions more tangible compared to the standard interfaces, with minor differences between the two-point interaction used in interface C1 and the drag-n-drop interaction in interface C2, which was however not significant. For the eight participants that effectively used the pinch/spread gestures, namely to zoom either the entire web site in interfaces A2 and B or specifically the picture in interfaces C1 and C2, it can be said that the adapted zoom to view a larger version of the picture was appreciated and rated higher. Still, due to the relatively large screen used in the study, the adapted zoom was generally not so often requested by participants and further studies on small-form factor devices should therefore aim to update these results. For other aspects, e.g. the almost vertical position of the touch display or technical limitations, such as limited precision and number of touch points that can be recognised with the TouchSmart, participants were generally neutral.

5 Observations and Implications for Design

The relatively high ratings for the user experience of the touch enhanced and multi-touch interfaces generally support our design decisions. One of the key factors that contributed to the fast times using interface A is that precise selection was not an issue with the mouse. Some participants even stated that the interface was optimally designed for mouse input because of the short distances between target elements. On the contrary, using the same interface on the touch display, participants had to concentrate on very precise selection and, for the nearly vertical setup that we used, often expected the touch to be recognised much higher than it was. Hence, we could often observe that participants developed a sort of counter technique in generally touching the screen slightly above the targets they actually wanted to hit, but this usually required some time to get used to. This was most often observed with activating the "Previous" and "Next" links or when trying to directly check the boxes associated with the listed tags rather than selecting the names. For the simple touch enhancements that we applied to the main navigation controls as well as the list of tags, the number of times participants missed the intended target elements on average were effectively reduced from 8 in the standard interface A2 using touch to a consistent .08 in all touch-enhanced interfaces B, C1 and C2—a significant improvement. As a result, the touch enhancements were sufficient to counteract some issues related to the precise selection of content.

We have already mentioned the fact that users rated the gestures for navigating between pictures relatively high. It also happened that users switched between techniques by sometimes using gestures and sometimes referring back to the "Previous" and "Next" links. In particular, participants found it faster to use the links when the tagging interaction previously occurred close to them. Web designers should therefore think of employing gestures as an alternative way of interacting with web content, not to completely replace standard means for interaction. This is especially important when users are already familiar with the traditional interface on non-touch devices and frequently switch between versions depending on the device in use, as for example in the case of Facebook.

6 Moving Forward

The study presented in this paper was driven by the current need to adapt interfaces for the emerging forms of multi-touch devices often used for web browsing. We have not only demonstrated that basic adaptations for touch can already contribute to better user experience, but also that it seems beneficial to further adapt interfaces to multi-touch interaction. In particular, we found it practical to start by addressing the low-level design issues first, such as the appropriate size and position of touch areas, and then address any issues with the existing interaction model as it is translated to a multi-touch design. We have already started to operationalise the key adaptation techniques used in this paper. Recently, we have added initial support in a multi-touch web interface toolkit, jQMultiTouch [1], and successfully used them as the basis for W3Touch [2], a metrics-based interface adaptation tool for touch. Our ongoing investigations have revealed several shortcomings of current web standards. Media queries provide a foundation for responsive design but, for the adaptations presented in this paper, they were not sufficient. One issue is that not all device aspects can be queried. For example, whether the TouchSmart was configured for touch rather than mouse input cannot be detected. Also information on the number of touch points supported by the device in use, namely two on the TouchSmart, is not available. Given that the latest proposals for CSS4 media queries cover only a few interaction media features, namely pointer and hover, this may not change in the near future[3]. In this regard, we want to critically note the remaining problem that state-of-the-art web technologies still lack common concepts and vocabulary, let alone a unified method, for the specification of multi-device web applications. In a related project, we have therefore investigated ways of enhancing existing languages with powerful context-adaptive mechanisms [10]. We would hope that similar concepts will make it to the web standards and be natively and consistently supported in future web browsers.

References

1. Nebeling, M., Norrie, M.C.: jQMultiTouch: Lightweight Toolkit and Development Framework for Multi-touch/Multi-device Web Interfaces. In: Proc. EICS (2012)
2. Nebeling, M., Speicher, M., Norrie, M.C.: W3Touch: Metrics-based Web Page Adaptation for Touch. In: Proc. CHI (2013)
3. Nebeling, M., Norrie, M.C.: Responsive Design and Development: Methods, Technologies and Current Issues. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 510–513. Springer, Heidelberg (2013)
4. Ivory, M., Megraw, R.: Evolution of Web Site Design Patterns. ACM Trans. on Information Systems 23(4) (2005)
5. Nebeling, M., Matulic, F., Norrie, M.C.: Metrics for the Evaluation of News Site Content Layout in Large-Screen Contexts. In: Proc. CHI (2011)
6. Findlater, L., McGrenere, J.: Impact of screen size on performance, awareness, and user satisfaction with adaptive graphical user interfaces. In: Proc. CHI (2008)
7. Hattori, G., Hoashi, K., Matsumoto, K., Sugaya, F.: Robust Web Page Segmentation for Mobile Terminal Using Content-Distances and Page Layout Information. In: Proc. WWW (2007)
8. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi- Target User Interfaces. IWC 15 (2003)
9. Paternò, F., Santoro, C., Spano, L.: MARIA: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments. TOCHI 16(4) (2009)
10. Nebeling, M., Grossniklaus, M., Leone, S., Norrie, M.C.: XCML: Providing Context-Aware Language Extensions for the Specification of Multi-Device Web Applications. WWW 15(4) (2012)

[3] http://dev.w3.org/csswg/mediaqueries-4

Composing JSON-Based Web APIs

Javier Luis Cánovas Izquierdo and Jordi Cabot

AtlanMod, École des Mines de Nantes – INRIA – LINA, Nantes, France
{javier.canovas,jordi.cabot}@inria.fr

Abstract. The development of Web APIs has become a discipline that companies have to master to succeed in the Web. The so-called API economy is pushing companies to provide access to their data by means of Web APIs, thus requiring web developers to study and integrate such APIs into their applications. The exchange of data with these APIs is usually performed by using JSON, a schemaless data format easy for computers to parse and use. While JSON data is easy to read, its structure is implicit, thus entailing serious problems when integrating APIs coming from different vendors. Web developers have therefore to understand the domain behind each API and study how they can be composed. We tackle this issue by presenting an approach able to both discover the domain of JSON-based Web APIs and identify composition links among them. Our approach allows developers to easily visualize what is behind APIs and how they can be composed to be used in their applications.

1 Introduction

The use and composition of different APIs is in the basis of computer programming. Software applications have largely used APIs to access different assets such as databases or middleware. In the last years, a new economy based on APIs has been emerging in the web field. To be competitive, companies are not only providing attractive websites but also useful Web APIs to access their data. Web developers have therefore to cope with the existing plethora of web APIs in order to create new web applications.

More and more web APIs use the JavaScript Object Notation (JSON) to exchange data (more than 47% of the APIs included in ProgrammableWeb[1] return JSON data). JSON is a schemaless data format easy for computers to parse and use. While JSON data is easy to read, its structure is implicit, thus entailing serious problems when integrating APIs coming from different vendors. In order to integrate external JSON-based web APIs, developers have to deeply analyze them in order to understand and manage the JSON data returned by their services. After analyzing JSON-based web APIs individually, it is still required to identify how to map the data coming from an API to call others since their implicit structure can differ.

Some approaches have appeared to make easier the understanding of JSON-based APIs, but they are still under development (e.g., RAML[2]) or are not widely used (e.g., JSON Schema[3] or Swagger[4]). Furthermore, the support for easily identifying how

[1] http://www.programmableweb.com
[2] http://raml.org
[3] http://json-schema.org
[4] http://swagger.wordnik.com

JSON-based web APIs can be composed is still limited. We believe that an approach intented to help developers to both understand and compose JSON-based web APIs would be a significant improvement.

In a previous work [1] we shown how to discover the schema which is implicit in JSON data. In this paper we build on that contribution to study how schemas coming from different JSON-based web APIs can be composed. Thus, we present an approach able to identify composition links between schemas of different APIs. This composition information plus the API schemas are used to render a graph where paths represent API compositions and are used to easily identify how to compose the APIs. For instance, we illustrate one application based on generating sequence diagrams from graph paths, where the diagram includes the API calls (and their corresponding parameters) that web developers have to perform in order to compose one or more APIs.

The paper is structured as follows. Section 2 motivates the problem. Sections 3, 4 and 5 describe our approach to discover the domain and composition links among JSON-based web APIs, respectively. Section 6 illustrates how our approach can be used to compose JSON-based web APIs and Section 7 discusses additional applications. Section 8 presents the related work and finally Section 9 concludes the paper and describes further work.

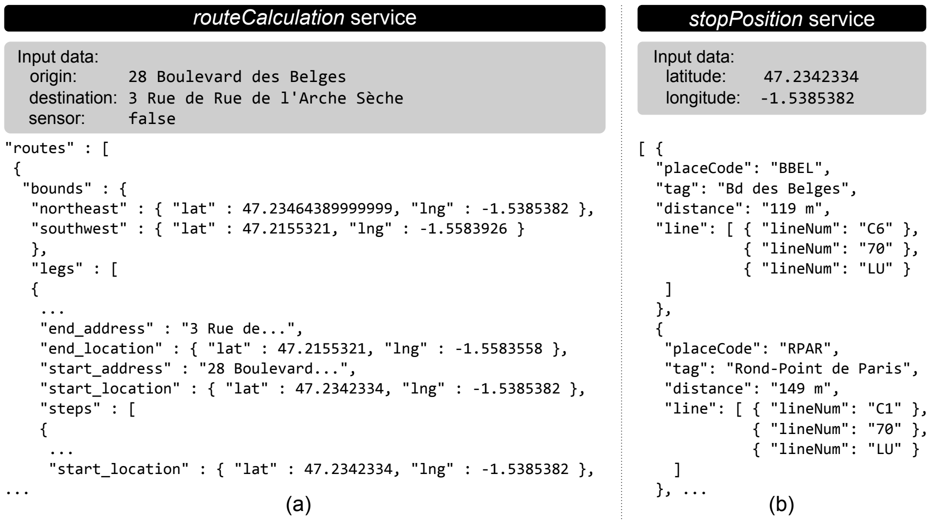
2 Using and Composing JSON-Based Web APIs

The development of web applications usually involves the composition of different web APIs. With the emergence of JSON-based APIs, web developers have to cope with the lack of documentation of these APIs and, when it exists, its non-standard format. Nowadays it is therefore usual to devote a significant amount of time to study JSON-based web APIs and to understand the implicit structure of the data they return. However, this is only the beginning since once APIs have been studied, it is required to explore how they can be composed (if possible). In this section we will show a simple example using two JSON-based web APIs we want to compose. From now on, we will refer JSON-based web APIs as APIs for the sake of conciseness.

Our example consists of a web application for tourists which includes a set of places to visit in our city and shows the routes to follow to reach them. The application includes a set of predefined places and needs to calculate the best route the user has to follow. Furthermore, the application also visualizes the bus/tram stops throughout the route, thus facilitating the route for old or handicapped people. Thus, we need two APIs to (1) calculate the best route between two points and (2) discover the bus/tram stops.

To calculate routes between points we will use the Google Maps API[5]. In particular, we will use the service to calculate the route to follow from a source point to a target one, which we will refer as *routeCalculation* service. This service receives as inputs: (1) the origin and (2) the destination of the route (expressed as addresses), and (3) whether a location sensor is available. The service returns a route to follow including the bounds and steps. Figure 1a shows an example of this service.

[5] `https://developers.google.com/maps`



routeCalculation service

```
Input data:
  origin:      28 Boulevard des Belges
  destination: 3 Rue de Rue de l'Arche Sèche
  sensor:      false
```

```
"routes" : [
 {
  "bounds" : {
   "northeast" : { "lat" : 47.23464389999999, "lng" : -1.5385382 },
   "southwest" : { "lat" : 47.2155321, "lng" : -1.5583926 }
   },
   "legs" : [
   {
    ...
    "end_address" : "3 Rue de...",
    "end_location" : { "lat" : 47.2155321, "lng" : -1.5583558 },
    "start_address" : "28 Boulevard...",
    "start_location" : { "lat" : 47.2342334, "lng" : -1.5385382 },
    "steps" : [
    {
     ...
     "start_location" : { "lat" : 47.2342334, "lng" : -1.5385382 },
...
```

(a)

stopPosition service

```
Input data:
  latitude:   47.2342334
  longitude:  -1.5385382
```

```
[ {
  "placeCode": "BBEL",
  "tag": "Bd des Belges",
  "distance": "119 m",
  "line": [ { "lineNum": "C6" },
            { "lineNum": "70" },
            { "lineNum": "LU" }
   ]
  },
  {
   "placeCode": "RPAR",
   "tag": "Rond-Point de Paris",
   "distance": "149 m",
   "line": [ { "lineNum": "C1" },
             { "lineNum": "70" },
             { "lineNum": "LU" }
    ]
  }, ...
```

(b)

Fig. 1. Two API calls examples: (a) the *routeCalculation* service from the Google Maps API and (2) the *stopPosition* service from the TAN API. The input data is shown on top while the resulting JSON is listed below. For the sake of clarity, the resulting data is shown partially and strings of the TAN API have been translated into English.

As we plan to deploy our example application in Nantes, France, we will use the API provided by TAN[6], the transportation entity of the city of Nantes, to discover the bus/tram stops along the calculated route. In particular, we will use the service we call *stopPosition*, which allows knowing the set of bus/tram stops near a given location. The service receives a position determined by the latitude and longitude, and returns the nearest tram/bus stops. Figure 1b shows an example of this service.

In this example the developer must first explore these APIs and then study how they can be composed (if possible). The analysis of the inputs/outputs allows identifying the main concepts used in each API (i.e., the domain). For instance, *routeCalculation* uses addresses to specify both the origin and destination of the route. Regarding its output data, locations are represented by `lat` and `lng` to specify the latitude and longitude, respectively. On the other hand, *stopPosition* receives a location as input and returns a set of bus/tram stops. After this study, the developer may come up with a possible mapping involving the values representing locations in the output of *routeCalculation* and the input of *stopPosition*, thus enabling their composition.

As can be seen, composing JSON-based web APIs require deeply studying the involved APIs and also how to compose them, which is a time-consuming and hard task, in particular, when dealing with a number of candidate APIs. In the remainder of this paper we will show our proposal to identify the domain behind APIs as well as data mappings which can help developers to easily compose them.

[6] `http://data.nantes.fr/donnees/detail/info-trafic-temps-reel-de-la-tan`

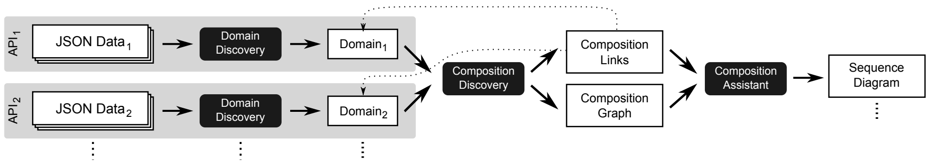


Fig. 2. Overall view of our approach. The main phases are represented with black-filled rounded boxes while input/output data is represented with white-filled boxes.

3 Our Approach

We propose an approach to study the composition of JSON-based web APIs. Our approach applies a discovery process which first analyzes the domains behind each involved API and then identifies composition links among them. The discovered information is used to render a graph in which calculations can be made to assist developers to compose APIs (e.g., sequence diagrams can be generated). Figure 2 illustrates our approach including the main two discovery phases (i.e., *Domain Discovery* and *Composition Discovery*) and facilities to realize the composition (see *Composition Assistant*).

Our approach represents domain information as class diagrams, including concepts (i.e., classes) and their relationships (i.e., attributes/associations), while composition links will be represented as relationships between concepts from different domains. We will leverage on model-driven techniques to represent both the domain and composition information as models and model references, respectively. The following sections will describe the main phases of our approach.

4 Domain Discovery in JSON-Based Web APIs

The domain of an API can be discovered by merging the domain of its services, which in turn can be discovered by analyzing the JSON data used as input/output. We devised a two-phase process to obtain the API domain represented as a model [1], which has been extended and adapted to enable the subsequent composition discovery phase (i.e., enriching the generated metadata). Next, we describe the basis of the process to facilitate the understanding of the remainder of the paper[7].

The first phase, called *Single-service discovery*, analyzes each service in order to discover its domain. Since JSON-based API services do not necessarily return JSON data conforming to the same structure, the accuracy of this phase increases when a number of JSON examples are provided. Thus, the single-service discovery phase analyzes a set of JSON examples (including inputs/outputs defined as JSON data) per API service. This phase is launched for each API service and has two execution modes: creation, which initializes the model concepts from JSON objects representing new concepts; and refinement, which refines existing model concepts with information coming from new JSON objects representing such concepts. Both execution modes are driven by a set of mapping rules transforming JSON elements into model elements[7]. As result, the single-service discovery phase returns a model representing the service domain.

[7] A detailed description of the process and mapping rules applied can be found in [1].

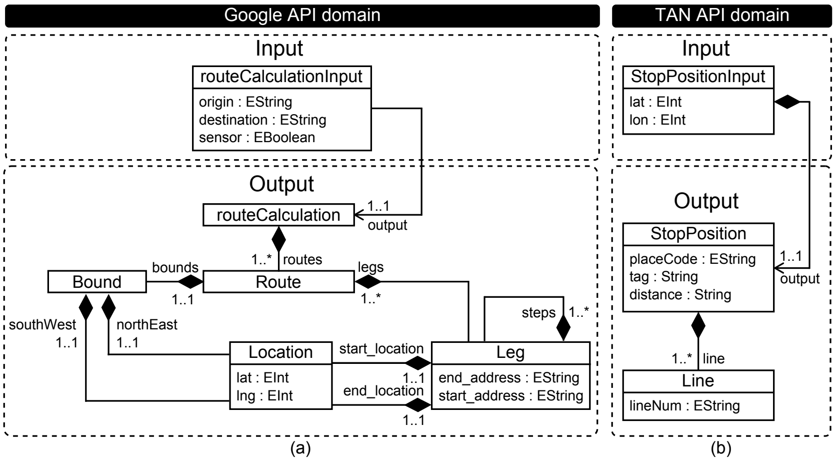


Fig. 3. Discovered domain for (a) Google API (including *routeCalculation* service) and (b) TAN API (including *stopPosition* service)

The second phase, called *Multi-service discovery*, composes the models generated by the previous phase and produces a new model representing the overall domain of the API. Similarly to what the single-service discovery phase does, several mapping rules are applied to obtain the composed model[7].

Figure 3 shows the API domains for the Google Maps and TAN APIs. For the sake of conciseness, we only show the excerpt of the model regarding the data shown in Figure 1. Note that since some JSON name/value pairs represent the same information, some concepts have been merged (e.g., the `Location` concept represents `northeast`, `southwest`, `end_location` and `start_location` JSON objects).

5 Composition Discovery in JSON-Based Web APIs

Composition links among APIs are discovered by means of matching concepts among their domains and analyzing whether they are part of the input parameters of API services. In this section we describe how to identify matching concepts and create composition links. These links can be later digested to facilitate the composition of the involved APIs, as we will explain below.

The discovery process of composition links analyzes the API domains to discover differences and similarities. However, this is not an easy task when dealing with models since the problem can be reduced to the problem of finding correspondences between two graphs (i.e., an NP-hard problem [2]). Based on our experience, we have identified a set of core rules but they can be extended by implementing other existing approaches (e.g., the ones presented in [3]):

R1 Two domain concepts *c1* and *c2* contained in different API domains are considered the same concept if *c1.name* = *c2.name*.

R2 As an API domain concept can represent several JSON objects (e.g., `Location` in Figure 3), only concept attributes/references found in every object are considered.

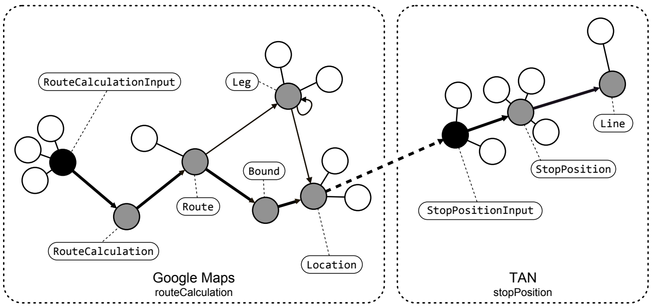


Fig. 4. Composition graph for the *routeCalculation* and *stopPosition* services

R3 Two attributes/references *a1* and *a2* are similar if *a1.name* = *a2.name* and *a1.type* = *a2.type*. Otherwise heuristics based on their name/type may be applied (e.g., the number of matching letters in their names must be higher than a given threshold).

R4 Two domain concepts *c1* and *c2* contained in different API domains are similar if they contain a number of similar attributes/references higher than a given threshold.

R5 There is a composition link between two domain concepts *c1* and *c2* contained in different API domains if they are the same (or similar) and *c2* is an input concept. The source of the composition link will be *c1* and the target will be *c2*.

The application of rules to our example will result in only one composition link from `Location` to `StopPositionInput` since R2, R3, R4 and R5 are fulfilled.

Composition links plus the API domains can be used to render a graph where nodes represent concepts/attributes and edges represent composition links or attribute composition. Figure 4 shows an example of this graph representation for our example. For the sake of clarity, nodes have been annotated with the name of the concept they represent. Gray-filled nodes represent the concepts used in each API, black nodes the concepts used as input to call an API, and white nodes the concept attributes, which are linked to the concept by an un-directed edge. Nodes are connected by directed edges, which can link nodes from the same (filled arrow) or different (dashed arrow) APIs. Nodes from the same API are linked when there is a reference between them, whereas nodes from different APIs are linked when a composition link has been detected.

6 Assisting Developers to Compose APIs

Paths in the graph can be used to assist developers in the composition of APIs. To calculate a path, developers must specify both the input information (by selecting the concepts/attributes they have available) and what they want to get (by selecting the desired concepts/attributes). Well-known graph algorithms can then be applied to calculate paths (if exist) among the selected nodes (through the directed edges). For instance, in our example we provide the attributes of the node *RouteCalculationInput* and our target node is *Line*. A possible path between these two nodes is highlighted in Figure 4, which indicates that a composition between these two APIs is possible. In particular,

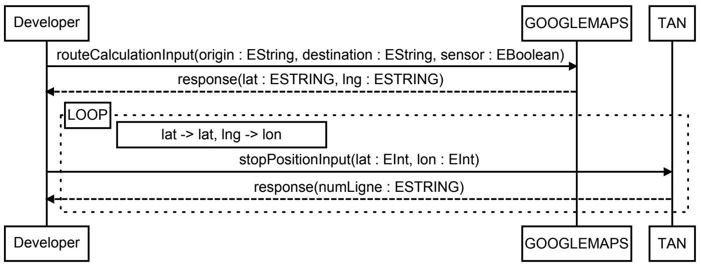

Developer
GOOGLEMAPS
TAN
routeCalculationInput(origin : EString, destination : EString, sensor : EBoolean)
response(lat : ESTRING, lng : ESTRING)
LOOP
lat -> lat, lng -> lon
stopPositionInput(lat : EInt, lon : EInt)
response(numLigne : ESTRING)
Developer
GOOGLEMAPS
TAN

Fig. 5. Sequence diagram generated from a path between `RouteCalculationInput` and `StopPositionInput` nodes of the graph shown in Figure 4

the composition can be performed calling the *RouteCalculation* service and using the attributes of the resulting `Location` concept to call to the *stopPosition* service.

Given this graph and the API domain models, several calculations can be applied to make easier the composition of the involved APIs and the understanding of paths in the graph. For instance, a sequence diagram can illustrate the calls and parameters to realize the composition. Figure 5 shows the sequence diagram for our example. Sequence diagrams can be drawn following these rules:

- There are as many actors as APIs are traversed by the path plus the developer actor.
- The diagram includes as many synchronous calls as APIs are traversed by the path.
- A method call is included for each API crossed. The method calls is named as the first node of the sub-path traversing the API and the parameters are its attributes. The method returns the set of attributes of the ending node of the sub-path.
- If the sub-path traverses a multivalued reference, the call for such path is a loop.
- A mapping between the output/input parameters of intermediate calls may be provided as annotation following the rules explained in Section 5.

7 Additional Applications

In previous sections we used a simple example to illustrate our approach and how paths in the graph can facilitate the composition of APIs. In this section we will increase the scope and size of the graph in order to study additional applications. In particular, we will focus on a cost-aware composition mechanism and obtaining the minimal branching subgraph. We will show first the extended graph we will use and then we will describe these applications.

Figure 6 shows the composition graph obtained from three real JSON-based web APIs, namely: Google Maps, TAN and an adapted version of the Foursquare API. Foursquare[8] is a social network allowing users to share their experiences when visiting places. For the sake of conciseness, we do not present the real name of each service but we will use an identifier[9]. Thus, the Google API includes four services (i.e., G_1,

[8] `http://foursquare.com`

[9] The graph can be found at `https://github.com/atlanmod/json-discoverer/tree/master/fr.inria.atlanmod.json.discoverer.zoo/exampleThreeAPIs`

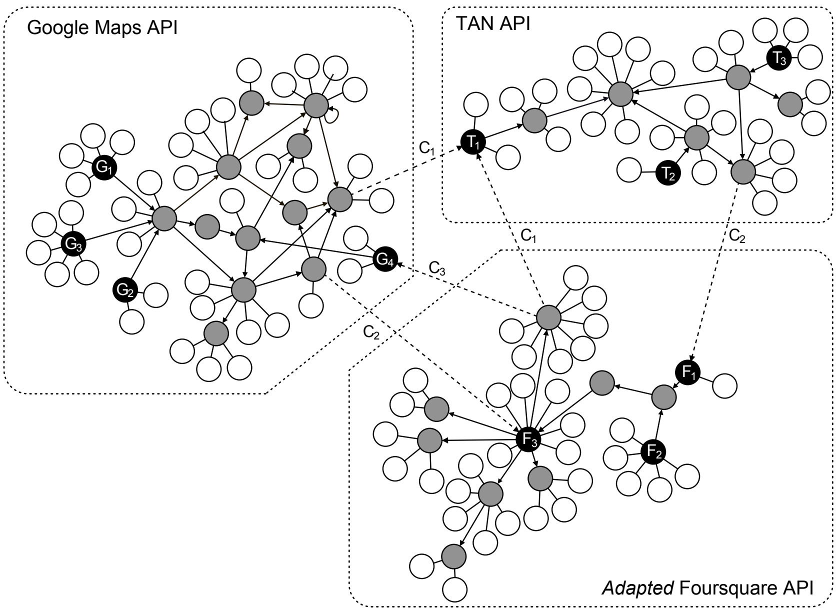


Fig. 6. Composition graph obtained from the services provided by three JSON-based APIs, namely, Google Maps, TAN and an adapted version of the Foursquare API

G_2, G_3 and G_4), the TAN API includes three services (i.e., T_1, T_2 and T_3) and the Foursquare API includes three services (i.e., F_1, F_2 and F_3).

Cost-Aware Composition. Some APIs follow a pay-per-use schema, e.g., fixed price per call, special price according to agreements, etc. To enable cost calculation, edges connecting different APIs can be annotated with the cost value. This information can then be used to obtain the best path (e.g., the cheapest path) among APIs.

Figure 6 includes annotations with the cost value in those edges connecting differente APIs. Thus, calling the TAN, adapted Foursquare and Google APIs costs C_1, C_2 and C_3 respectively. A possible scenario could be as follows. As described before, developers can compose the Google and TAN APIs by means of the services G_1 and T_1, which costs C_1. However, there exists a second option which involves composing the three APIs of the graph (i.e., the path will start with the G_1 node of the Google API, then will cross the F_3 node of the adapted Foursquare API and finally the T_1 node of the TAN API), which costs $C_2 + C_1$. Depending on the concrete values of these costs, developers can decide which one is the most suitable to their needs.

Minimal Subgraph. The graph shown in Figure 6 also allows developers to discover all the composition paths among the analyzed APIs. In order to facilitate the identification of all the API compositions, it is possible to apply traditional graph algorithms to calculate the optimum branching (such as [4]), which will provide the minimal path among every node of the graph. The developer can also prune some nodes and recalculate the graph in those cases in which a path crosses some nodes representing concepts/attributes that the developer cannot provide.

It is important to note that composition paths may not access to every API node since the API subgraph may not be a strongly connected graph. For instance, the composition of the Google API with the TAN API provides access to the latter API through the service T_1, which is connected with a limited number of nodes of the TAN subgraph. Thus, when the API subgraph is not a strongly connected graph, to be precise, composition paths should be indicated in terms of API services.

8 Related Work

The discovery of the implicit structure in JSON data is related to works focused on obtaining structured information from unstructured data such as [5]. Our approach integrates some of their ideas. Furthermore, the use of metadata in the model discoevry phase has been inspired by works such as [6, 7].

Composition link discovery applies some basic tecniques to detect matching modeling elements. Several works such as [8–11] and tools such as EMFCompare [12] could be used here to improve our discovery process.

In the field of web engineering, our approach is related to those ones focused on web services. For instance, [13] proposes an approach based on semantic web services which are analyzed to discover how they can be coreographed (i.e., composed). A similar approach to ours has been presented in [14], where a solution to integrate and query web data services is presented. The approach resorts in the web service definitions (i.e., WSDL) to define *service interface*s which are later analyzed to discover possible ways to integrate and query them. The Yahoo Query Language (YQL)[10] is also related to our approach, since they allow perfoming queries among web services with the aim of composing them. Finally, in the particular field of mashups, the works [15, 16] also address the problem of composing different web services. Regarding the data used, the main difference with these approaches is that ours is specifically adapted to deal with JSON-based web APIs, where generally there are no formal definitions of the services (as it could happen with web services by means of definition such as WSDL or semantic web services with OWL-S). However, with regard to the mechanisms to discover potential composition links, our approach can be enriched adapting their proposals.

9 Conclusion and Future Work

In this work we have presented an approach to study how JSON-based web APIs can be composed. Our approach leverages on a previous work and extends it to infer composition links among APIs. Composition information is represented as graphs, where paths represent concrete API compositions. Furthermore, these paths are used to create sequence diagrams to facilitate the understanding of the composition. Our tool has been fully implemented and is available as a free service[11].

[10] `https://developer.yahoo.com/yql`
[11] `http://atlanmod.github.io/json-discoverer`

As future work, we plan to explore other possible ways to facilitate API composition, such as generating the glue code among them. We would also like to study new mechanisms to detect concept similarities (e.g., using WordNet[12]) as well as conduct a quantitative evaluation of our approach to study its scalability.

References

1. Cánovas Izquierdo, J.L., Cabot, J.: Discovering Implicit Schemas in JSON Data. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 68–83. Springer, Heidelberg (2013)
2. Lin, Y., Gray, J., Jouault, F.: DSMDiff: a differentiation tool for domain-specific models. Europ. Inf. Syst. 16(4), 349–361 (2007)
3. Kolovos, D.S., Di Ruscio, D., Pierantonio, A., Paige, R.F.: Different models for model matching: An analysis of approaches to support model differencing. In: CVSM Conf., pp. 1–6 (2009)
4. Edmonds, J.: Optimum Branchings. J. Res. Nat. Bur. Standards 71B, 233–240 (1967)
5. Nestorov, S., Abiteboul, S., Motwani, R.: Inferring structure in semistructured data. ACM SIGMOD Record 26(4), 39–43 (1997)
6. Famelis, M., Salay, R., Di Sandro, A., Chechik, M.: Transformation of Models Containing Uncertainty. In: Moreira, A., Schätz, B., Gray, J., Vallecillo, A., Clarke, P. (eds.) MODELS 2013. LNCS, vol. 8107, pp. 673–689. Springer, Heidelberg (2013)
7. Famelis, M., Salay, R., Chechik, M.: Partial models: Towards modeling and reasoning with uncertainty. In: ICSE Conf., pp. 573–583 (2012)
8. Alanen, M., Porres, I.: Difference and union of models. In: Stevens, P., Whittle, J., Booch, G. (eds.) UML 2003. LNCS, vol. 2863, pp. 2–17. Springer, Heidelberg (2003)
9. Ohst, D., Welle, M., Kelter, U.: Differences between versions of UML diagrams. In: ACM SIGSOFT Conf., pp. 227–236 (2003)
10. Selonen, P., Kettunen, M.: Metamodel-Based Inference of Inter-Model Correspondence. In: CSMR Conf., pp. 71–80 (2007)
11. Melnik, S., Garcia-molina, H., Rahm, E.: Similarity Flooding: A Versatile Graph Matching Algorithm. In: DE Conf., pp. 117–128 (2002)
12. Brun, C., Pierantonio, A.: Model Differences in the Eclipse Modeling Framework. UPGRADE, The European Journal for the Informatics Professional 9(2), 29–34 (2008)
13. Sycara, K.P., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated discovery, interaction and composition of Semantic Web services. J. Web Sem. 1(1), 27–46 (2003)
14. Quarteroni, S., Brambilla, M., Ceri, S.: A bottom-up, knowledge-aware approach to integrating and querying web data services. TWEB 7(4), 19 (2013)
15. Daniel, F., Rodríguez, C., Chowdhury, S.R., Nezhad, H.R.M., Casati, F.: Discovery and reuse of composition knowledge for assisted mashup development. In: WWW Conf., pp. 493–494 (2012)
16. Chowdhury, S.R., Daniel, F., Casati, F.: Efficient, interactive recommendation of mashup composition knowledge. In: ICSOC Conf., pp. 374–388 (2011)

[12] `http://wordnet.princeton.edu/`

Design Criteria for Web Applications Adapted to Emotions

Giulio Mori, Fabio Paternò, and Ferdinando Furci

ISTI – CNR, Via Moruzzi, 1, 57126 Pisa, Italy
{giulio.mori,fabio.paterno,ferdinando.furci}@isti.cnr.it

Abstract. The main goal of this work is to identify a set of design criteria for Web applications taking into account the users' emotions. The results are based on the analysis of a user study with 50 participants who tested six Web interfaces, each one designed to elicit a specific emotion (hate, anxiety, boredom, fun, serenity, love). The design criteria applied to the six emotion-based Web interfaces were drawn from the results of a previous survey, which involved 57 different users, on the relationships between emotional state and Web interfaces. This initial survey asked the users to indicate the emotions most often associated with Web interaction, and then assign each emotion with some specific Web design characteristics. The resulting design criteria can form the basis for a set of emotion-related guidelines for Web application user interfaces.

Keywords: Web guidelines, emotions, affective interfaces.

1 Introduction

The important role of emotions in HCI is widely accepted [3-5]. However, little work has been dedicated to how to take them into account in Web applications, which are the most widely used applications. Thus, Web developers and designers need support on some design criteria (such as choice of user interface elements, navigation style, suitable colors, etc.) associated to emotional states. Emotions are complex and depend on individual preferences, attitudes, moods, affect dispositions, and interpersonal stances; "there is no single standard gold-method for their measurement" [10].

The literature reports different methodological approaches to classifying emotions, such as Geneva [8] or Feeltrace [9], and scales and questionnaires to measure either two primary (negative-positive) dimensions of moods [1], or hedonic and pragmatic dimensions of user-experience [7][12], but no work has focused on typical emotions during Web interaction. There are some works comparing different versions of Web pages [5] to investigate the impact of their attractiveness [2] or aesthetics, or if one Web site is better than another at eliciting emotions [4]. An analysis of existing Web sites about the hedonic elements (such as color, images, shapes and photographs) has been carried out [11] to investigate the emotional appeal, the sense of the aesthetic or positive impression resulting from the overall graphical look of a Website. However, none of these studies provide precise indications on how the various aspects of Web interfaces can elicit a specific emotion. Modeling the key subjective aspects of user experience and how it affects the perception of the final product or emotional responses [4] are important contributions, but there is the need to better address the emotion-based Web design aspects.

In this paper we aim to investigate the impact of some Web design criteria to elicit a particular emotional state on the user, independently of the application domain. In order to better understand what design choices are most suitable for most recurrent user emotions in Web applications, we have conducted two user studies described herein. Since there were no specific indications in the literature about the effects of Web design on eliciting a specific emotion, we organized a first survey to start our research, with the goal of collecting some basic indications from a sample of 57 users. The survey aimed to better understand the most recurrent emotions during Web interaction and the related Web design features. Next, we wanted to check the effectiveness of the data gathered in the survey through a user study. We applied the criteria to six Web interfaces, each one designed to elicit a specific emotion. Fifty different users judged their emotional impact. Positive emotions are certainly important for improving the user experience, however, also understanding the Web design criteria eliciting negative emotions has its importance for Web designers in order to improve their awareness of the risks and kinds of emotions their Web sites may elicit. In particular, section 2 describes the initial survey having the goal to investigate the relationships between emotional state and Web interfaces. Section 3 reports on the user study having the goal to validate the design principles through six Web Interfaces, and finally, we draw some conclusions and provide indications for future work.

2 A Survey: Collecting Opinions about Emotional Web Design

The questionnaire was completed by 57 users in two sessions (the average completion time per user was about two hours): the first session was carried out in the presence of the authors (to provide the users with explanations, when necessary), and a second session in which the users completed the questionnaire alone. The questionnaire was composed of three parts: 1) personal information, 2) classification of emotions in Web interaction, 3) emotion-based Web design opinions. The first part aimed to collect some personal information from the users and their experiences with the Web. In the second part, users had to propose some emotions they considered relevant during Web interaction and, for each of them, they had to freely associate colours and some attributes characterizing the user activity. The third part was more oriented to the Web design, where users had to give their opinions by associating each emotion with various Web interface features (depending on the question, we showed them various graphical examples of typical elements of a Web interface).

2.1 Personal Information

The participants were 25 females and 32 males, with an average age of 38,21 years (ranging from 26 to 59). Seventeen users had a PhD, 21 users had a five-year degree, 2 users had a four-year degree, 8 users had a bachelor's degree, while 7 users had a high school diploma and 2 users had a school diploma. Users were used to surf the Internet (46 users were connected to the Web every day, 9 users navigated three times per week, 2 users used the Web one time every fifteen days). The sample considered both experienced and inexperienced users in Web development (at different levels, 38 users had implemented some Web interfaces, while 19 users had little or no knowledge on Web programming).

2.2 Classification of Emotions in Web Interaction

In this part, each user had to indicate a certain number of emotions (maximum 8) which s/he considered relevant during the Web interaction. The only constraint of their suggestions was that for each chosen emotion, they also had to indicate the opposite emotion (depending on the emotional valence, negative or positive, in their perception). The reason for this request was that we wanted to define a complete design space that given a negative emotion could allow the identification of the positive counterpart. At the beginning of the questionnaire, some users found difficulties to define emotions and needed support, however at the end, the total average number of the proposed emotions was 3.84 per user, with a total of 219 distinct proposed emotion names perceived as negative and other 219 perceived as positive emotions. Each user had to associate a value (in an ordered scale from -8 to +8) to each emotion, as a measure to indicate how negative or positive s/he considered it. Analyzing the proposed emotions of the 57 users, we decided to consider as more "significant" only the emotions with at least 10 preferences, discarding the remaining others, which had received just only 1, 2 or at maximum 3 preferences each one. In addition, after having analyzed the complete final results of the questionnaire, we noticed that some emotions (having different proposed names by the users, but similar meaning), were characterized by the same Web features. In these cases, with the consent of the users who proposed the emotions, synonyms have been considered the same emotion, and finally, we have obtained an essential basic ordered scale of 6 emotions (3 negative and 3 positive) to express the typical affective states of a user interacting with Web (see Table 1), corresponding to well distinguishable Web design characteristics. The final emotions would have been the same also first joining the synonyms and then filtering the emotions with at least 10 preferences because in addition to the "primary emotions" [13] (such as hate, love, etc.) many users proposed a lot of different "secondary emotions" [13] (such as jealousy, nostalgia, loneliness, etc.), which were proposed only by few other users.

Considering the complexity of the emotions world for human beings and the many emotion classifications existing in literature [8][9][10] (even if no one is specifically oriented to the Web interaction and design), this classification is not exhaustive. However, the goal of this work is not to provide a further emotion classification, but rather to investigate whether some clearly distinguishable design characteristics can elicit a specific emotion on the user independently of the application domain. So this small starting set composed of 6 emotions oriented to Web interaction (obtained by the users suggestions) has been the basis of our study. The goal is to understand if some different Web design features can concretely have a specific impact on the users perception and on the personal emotional state, independently of the contents. Looking at Table 1, some considerations about the meanings of the emotions in the scale proposed by the users are necessary: a) *hate* and *love* express the sense of disliking/liking or indifference/empathy for something or somebody (typical of Web social network environments and Web 2.0). English language expresses well these meanings (i.e.: I hate/love Louis Armstrong music); b) *anxiety* and *serenity* express the emotional state during critical/safe actions (i.e.: the user is booking/buying something on Web and s/he need to fill a form inserting personal or credit card data); c) *boredom* or *fun* depends on the interest of users for the Web content, and the way the contents are presented is fundamental. Table 1 shows the ordered scale of the emotions with the corresponding average values proposed by the users.

Table 1. The main 6 emotions indicated by the users. The order is determined by the average values assigned by the users on the scale of a negative or positive personal perceptions.

Average values of the main Web Emotions considered relevant by the users					
Hate	Anxiety	Boredom	Fun	Serenity	Love
-3.48	-2.54	-1.99	+2.1	+2.25	+3.6
Standard Deviation values of the main Web Emotions considered relevant by the users					
1.29	1.04	0.76	0.92	0.83	1.23

In this part of the questionnaire, we also asked the subjects to associate some attributes related to the user activities for each emotion. Users had to select from the following pairs: static or dynamic (perceived level of changes of the interfaces reflecting the changes of the personal emotional state), passive or active (perceived level of the user's involvement in doing actions), simple or complex (perceived level of how the interface can be elaborated in design). The results are summarized in Table 2. As a criteria of choice (for the results of Table 2 and for the other tables presented for this first survey), we took into considerations an attribute as *strongly characterizing an emotion*, when the total number of users' preferences for a value of a pair (or group) was at least the double (over 50%) of the other/s preferred choices; in borderline cases, conclusions could be ambiguous. The double threshold was a prudential strategy, because we noticed that when a characteristic was chosen quite unanimously by most of the users, that characteristic collected a number of preferences higher than the double (in comparison with the other choices).

Table 2. User activity attributes for each emotion

Emotions	**User activities attributes**
Love	dynamic, active, complex
Serenity	static, simple
Fun	dynamic, active
Boredom	static, passive, simple
Anxiety	dynamic, complex
Hate	static, active

The following part of the questionnaire asked also to the users to associate one or more colors (preferably belonging to the 16 HTML standard color palette supported by all browsers [6]) to each emotion, and then it was asked to associate freely some visual characteristics, real objects or abstract ideas. The exact tint of the 16 colors were showed to each user during the questionnaire. Table 3 summarizes the results.

Table 3. Colours and visual characteristics for each emotion

Emotions	Main Colors	Visual Characteristics
Love	red, pink	bright, transparent, indefinite, heart
Serenity	blue, aqua, white, green, lime	clear, bright, calm waters, large open spaces, open sky, nature, flat, smooth, light colors
Fun	fuchsia, red, orange, yellow, green, lime, teal, aqua	brilliant, bright, colorful, spring, light, sun, flowers, light fire
Boredom	silver, gray, black	night, dark, blurred, indefinite, hazy, fog, opaque, rain, tears, dim, empty
Anxiety	black, gray, navy, yellow	night, dark, wavy, intermittent, storm, throbbing, blurred, indefinite, fog
Hate	black	night, dark

2.3 Emotion-Based Web Design

The last part of the questionnaire asked the users suggestions regarding specific associations between each emotion and many Web design features. This part aimed to investigate the structure, the multimedia elements and type of interactive elements being the most effective to evoke a specific emotional state. The users could express their preference about any elements of the interface or Web design features, (we showed for each question some graphical samples with the goal that each proposed choice was clear for the users). In particular, first the users had to choose between the following groups related to the Web site structure and the interaction elements for data insertion: 1) few or many pages, 2) blurred or clear text, 3) short or long text, 4) presence of textbox (to insert short data) or textarea (to insert long information, e.g. requests). Table 4 summarizes the results.

Table 4. Page contents & structure for each emotion

Emotions	Pages & Content structure
Love	few pages, long text, clear text, textarea
Serenity	little content, short text, clear text, textarea
Fun	many pages, short text, clear text, textbox
Boredom	few pages, long text, textbox
Anxiety	many pages, blurred text, textarea
Hate	few pages, blurred text, textbox

We then asked the users their opinions about the emotional impact of the multimedia element style, whereby the users had to choose one option from each group: 1) presence of video, animations, images, or no multimedia, 2) small, medium or large size of the images, 3) definition of the images (blurred or clear), 4) color, black & white, or de-saturated (the color was reduced) images. Table 5 shows the results. It is not trivial at all (we cannot say if it depends on some cultural factors or something unconscious in human beings), observing that every user suggested unanimously color and clear images for every positive emotion, and blurred and no color images for two negative emotions hate and anxiety (for boredom absence of images or video was perceived by the users as a factor more emphasizing boredom).

Table 5. Multimedia elements style for each emotion

Emotions	Multimedia Elements
Love	color medium/large clear images
Serenity	color medium-clear images, videos
Fun	animations, color medium-clear images, videos
Boredom	no images, no videos
Anxiety	de-saturated small blurred images, videos
Hate	black & white medium/large blurred images, videos

The questionnaire also sought to explore the users' opinions about the navigation elements (Table 6), choosing amongst the following options: 1) links, standard buttons, graphic buttons or tabs, 2) static or dynamic effects on navigation elements.

Table 6. Navigation elements for each emotion

Emotions	Navigation Elements
Love	graphic buttons, dynamic effects
Serenity	tabs, link, graphic/standard buttons
Fun	graphic buttons, dynamic effects
Boredom	links/standard buttons, static effects
Anxiety	standard/graphic buttons, dynamic effects
Hate	standard/graphic buttons, static effects

Finally, the users gave their opinions about the interactive elements (Table 7), choosing from among the following options: 1) interactive elements with static or dynamic effects 2) textual or graphic interactive elements, 3) radio-button or pull-down single selection, 4) checkboxes, scroll or fixed multiple selection.

Table 7. Interactive elements for each emotion

Emotions	Interactive Elements
Love	dynamic, graphic, pull-down, checkboxes/scroll selection
Serenity	static, textual
Fun	dynamic, graphic
Boredom	static, textual, checkboxes
Anxiety	dynamic, graphic, pull-down menu, scroll select
Hate	checkboxes

As a confirmation of the complexity of the emotions, it is not surprising that tables contain some overlapping Web features between the six emotions. With this survey we received indications about many other aspects of Web design (such as position of the elements, font type/dimension, alignment of text, etc.). We do not report these extra data, because the results are too ambiguous and further tests are necessary.

2.4 Additional Emotion-Related Design Aspects

In addition to the analysed data, we received interesting comments of the participants suggesting us useful indications to improve the emotional effect of Web design. Even if enumeration of these comments was not possible because some users did not provide any comment, we decided to apply them to the Web interfaces design (see section 3) to verify their effects. We report below the summary of the key indications:

Hate-Love. A user can feel *hate* due to the design of the interface, if the interface is difficult to use (bad usability), the layout of the interface and/or the positioning of the widgets are confused (not easy to understand). In the worst case, the interface obstacles the interaction of the user, or/and something is not working (i.e. elements of the interface are not responding to the user input). On the contrary, a user can feel *love* due to the design of the interface, if the appealing aesthetic of the Web interface stimulates its use. Besides, the interface should be easy to use (good usability), the layout and disposition of the widgets should be disposed in a way easy to understand.

Anxiety-Serenity. A user can feel *anxiety* due to the design of the interface, if the interface emphasizes particular stress factors (i.e. a deadline, risk, or sense of losing something, etc.). In these stress conditions, the interface does not allow the user to reason comfortably (i.e. adding intermittent light effects, distortion or jerky transformations of the elements in the interface, etc.). On the contrary, a user can feel *serenity* due to the design of the interface, if the interface let the user to feel safe, (i.e. providing always feedbacks, or showing well known reassuring elements, as a logo of secure transactions, etc.). Besides, the simplicity of the interface allows the user to interact easily, reducing her/his effort and giving the time s/he needs.

Boredom-Fun. A user can feel *boredom* due to the design of the interface, if the interface provides or requires lots of information (i.e. very long texts, or many required fields in a form, etc.). Much text without images or multimedia elements, increases boredom. On the contrary, a user can feel *fun* due to the design of the interface, if unexpected elements, animations or effects surprise the user in a positive way. The animations or dynamic effects should be oriented to facilitate the interaction, otherwise they are perceived as annoying.

3 A User Study: Applying the Emotion-Based Design Principles

On the basis of the results of the survey, we developed six Web interfaces, applying the collected Web designs principles. Each Web interface had the goal to elicit one of the six emotions of the scale. Each Web interface presented the same content (except very minimal additions suggested by the users) in a different design style. Considering that most users in the survey suggested music as topic for an emotion-based Web application, we chose the Beatles' musical history as a topic for the Web six interfaces. In details, the interfaces contained: a short textual biography, a media player to listen to five famous songs, a musical video, a form where the user could buy some virtual tickets for revival musical events, and some clickable graphical covers of six famous albums. Finally, we recruited 50 different users (through a mailing list of our institute), who had not participated to the previous survey, to test the six Web interfaces through some interactive tasks, and after that, to judge (through a questionnaire) the Web design effectiveness in stimulating a specific emotion.

3.1 Description of the User Test and Discussion of the Results

We showed the six Web interfaces to the 50 users in random order. We wanted to avoid that the order could influence the emotional perception. Each user had to test each Web interface by completing three tasks, and then s/he had to fill in an online questionnaire. The questionnaire was composed of three sections, and it asked the users: a) personal information, b) judgment about the effectiveness of the interface to stimulate the proposed emotion and comments, c) suggestions about other emotions we did not consider in the classification, opinions about the utility of the adaptation of the Web design to elicit more positive emotions on the users, and some suggestions about useful application fields. Most users considered the six emotions exhaustive for the Web interaction (even if some users proposed anger or surprise as examples of additional emotions). Nearly all the participants found the adaptation of the Web design to elicit positive emotions very useful. As important applications for applying the emotional Web design, they suggested educational environments, telemedicine and online psychological platforms, games, home automation applications, or tools oriented to helping people with disabilities or the elderly. The users preferred Web design stimulating positive emotions because it improves the user experience. However, they also considered the utility of eliciting negative emotions not only to improve the awareness of Web designers, or to recreate particular thrilling atmospheres in games, but also in the telemedicine field. In this area the ability to understand the reactions of patients in a good or bad affective state is important. Moreover, it could improve children's awareness about the difference between good and bad behavior in some educational or learning tools. The users gave their judgment in a scale from 1 to 5 (where the value 1 indicated that the page was very ineffective to elicit the proposed emotion, while value 5 indicated that the page was very effective, and the value 3 represented the neutrality). Considering that the survey was "open answer" (concerning the users' proposal of Web emotions), we decided for this user test that the users could know in advance the emotional goal of the currently tested interface. We wanted to avoid that another "open-answer user test" could produce too vague results. In particular for each emotion, the users were asked to accomplish three tasks: a) find the answer to a proposed question in the biography text (the goal was obliging the user to read the text with specific style characteristics), b) click on one proposed cover of one album (the goal was obliging the user to evaluate the interaction with the elements of navigation), c) fill in a form (with proposed data corresponding to one imaginary user) to buy some virtual tickets (the goal was obliging the user to evaluate the elements of interaction). The questionnaire was completed by 50 users (average completion time per user of about one hour). The participants were 21 females and 29 males, with an average age of 38,28 years (ranging from 26 to 77). Ten users had a PhD, 21 users had a five-year degree, 4 users had a four-year degree, 9 users had a bachelor's degree, while 6 users had an high school diploma. Users were used to surf the Internet (42 users were connected to the Web every day, 5 users navigated three times per week, 2 users used the Web one time every fifteen days, and one used Web when it happens). The sample considered both experienced and inexperienced users in Web development (at different levels, 28 users had implemented some Web interfaces, while 22 users had little or no knowledge on Web programming).

The six Web interfaces were designed as follows: a) the Web interface to elicit hate was completely unusable with a confused layout, b) the Web interface to elicit anxiety showed intermittent light effects and jerky transformations, with a countdown as a pressure factor to fill in the form, c) the Web interface to elicit boredom, was neutral, without images or videos, requiring more fields to fill in the form, d) the Web interface to elicit fun, showed unpredictable animations and dynamic effects, e) the Web interface to elicit serenity, was very simple to minimize the user's effort, f) the Web interface to elicit love had an appealing graphics and it was usable. For lack of space, we have to omit further details. The average judgment of the 50 users about the effectiveness of the interfaces to elicit the proposed emotion was high: (over value 4) for hate, anxiety, boredom and serenity, while it was slightly effective (over value 3) for fun and love. The results of the user test showed that the usability, even if it is an important factor, it is not the unique aspect responsible to elicit an emotional reaction on the user (e.g. the different unusable interfaces aiming to elicit hate and anxiety, produced different emotional effects). Even if the results are encouraging, a more detailed statistical analysis is necessary. In particular it is important to understand for each emotion, if a subset ("core") of key design factors can be responsible for eliciting a specific mood. It is necessary investigate further if the results can depend on some "confounding factors" (such as age, gender, experience in development, etc.), or if the results are domain/topic-dependent or domain/topic-independent.

4 Conclusions and Future Work

The goal of this work has been to study whether the collected preliminary user indications about the relevant emotions during Web interaction and the corresponding specific design criteria actually do have an emotional impact on users. The results obtained are encouraging, even if further refinements and investigation are necessary. In particular, other user tests will be necessary to better understand the essential Web design key factors affecting emotional state. The ultimate goal of this research is the formalization of a set of Web guidelines to design interfaces that can effectively stimulate user emotions during the interaction. For the implementation of the next user tests, we are considering various sensors to detect physiological parameters of the users in order to monitor their changes and to adapt the Web design with the goal of eliciting more positive emotions on the users.

References

1. Watson, D., Clark, L.A.: Development and Validation of Brief Measures of Positive and Negative Affect: the PANAS Scales. Journal of Personality and Social Psychology 54(6), 1063–1070 (1988)
2. Hartmann, J., Sutcliffe, A., De Angeli, A.: Investigating Attractiveness in Web User Interfaces. In: Proc. CHI 2007. ACM Press (2007)
3. Hassenzahl, M.: The Think and I: Understanding the relationship between user and product. Funology Human Computer Interaction Series 3, 31–42 (2005)
4. Karlsson, M.: Expressions, Emotions, and website design. CoDesign 3(1), 75–89 (2007)
5. Kim, J., Lee, J., Choi, D.: Designing Emotionally Evocative Homepages: An Empirical Study of the Quantitative Relations Between Design Factors and Emotional Dimensions. International Journal of Human-Computer Studies 56(6), 899–940 (2003)
6. The 16 HTML color names, `http://en.wikipedia.org/wiki/Web_colors`
7. Voss, K.E., Spangenberg, E.R., Grohmann, B.: Measuring the hedonic and utilitarian dimensions of consumer attitude. Journal of Marketing Research 40(3), 310–320 (2003)
8. Bànziger, T., Tran, V., Scherer, K.R.: The Geneva emotion wheel. Journal, Social Science Information 44(4), 23–34 (2005)
9. Cowie, R., Douglas-Cowie, E., Savvidou, S., McMahon, E.: Feeltrace: an instrument for recording perceived emotion in real time. In: Proceedings of the ISCA Workshop on Speech and Emotion 2000, pp. 19–24 (2000)
10. Scherer, K.R.: What are emotions? And how can they be measured? Social Science Information 44(4), 695–729 (2005)
11. Cyr, D.: Emotion and Website Design, 2nd edn., ch. 40., `http://www.interaction-design.org/encyclopedia/emotion_and_website_design.html`
12. Diefenbach, S., Hassenzahl, M.: The Dilemma of Hedonic – Appreciated, but hard to justify. Interacting with Computers 23(5), 461–472 (2011)
13. Damasio, A.R.: Descartes' error: Emotion, reason, and the human brain. Avon Books, New York (1997)

Driving Global Team Formation in Social Networks to Obtain Diversity

Francesco Buccafurri, Gianluca Lax, Serena Nicolazzo, Antonino Nocera, and Domenico Ursino

DIIES, University of Reggio Calabria
Via Graziella, Località Feo di Vito
89122 Reggio Calabria, Italy
{bucca,lax,s.nicolazzo,a.nocera,ursino}@unirc.it

Abstract. In this paper, we present a preliminary idea for a crowdsourcing application aimed at driving the process of global team formation to obtain diversity in the team. Indeed, it is well known that diversity is one of the key factors of collective intelligence in crowdsourcing. The idea is based on the identification of suitable nodes in social networks, which can profitably play the role of generators of diversity in the team formation process. This paper presents a first step towards the concrete definition of the above application consisting in the identification of an effective measure that can be used to select the most promising nodes w.r.t. the above feature.

1 Introduction and Description of the Idea

It is well known that diversity is one of the key factors of collective intelligence in crowdsourcing [20]. On the other hand, it is clear that this concept fully confirms the famous principle summarized as *the strength of weak ties*, stated in the field of social networks [14]. The two worlds, social networks and crowdsourcing, have a strong overlap, as social-network users form a huge crowd. But, social-network crowd includes something more than the simple Web crowd. It has a friendship-based structure, embeds contents, and is full of knowledge about people. This opens a lot of opportunities that can reinforce the power of crowdsourcing (e.g., see [16]). For instance, consider the problem of dynamic formation of globally distributed teams for enterprisers [21]. Driving team formation can result in tangible benefits for the success of the team work, as a number of features of the individuals, such as expertise, should be considered. Social networks are repositories of a large amount of information about people, in which we can find the aimed features. But this is not enough. Indeed, it is not what a crowdsourcing process, thus spontaneous and evolutionary, requires. As a matter of fact, individuals in a social networks are not monads. So, not only friendship relationships allow the autonomous flooding of the network, enabling crowd formation, but the type of ties on which the crowd formation propagates can be dramatically important for the final quality of the global team: We have to hope that the most of crossed ties are weak, to fully reach the goal of diversity. Therefore, the basic principle we are stating here is that global team formation in crowdsourcing, even though mostly spontaneous, should be driven in some way by taking advantage of social networks with the aim of maximizing diversity.

In this paper, we try to answer a simple question: How to translate the above principle in a social-web application aimed at improving the quality of global team formation in crowdsourcing? This is a preliminary research, as it contains an idea on how to do this, but it is focused on a very important aspect which is the starting point for transforming this idea into a concrete application. The idea is that team formation should be driven by weak ties, as mentioned earlier. But, how to find weak ties? From social network analysis we know that the concept of weak tie is related to behavioral aspects (for example, the number of interactions in the dyad), which are very difficult to capture in a feasible way. Actually, the concept of weak tie is also related to structural properties, since weak ties are typically bridges between two different communities. In particular, it is related to centrality measures [11]. Among these, betweenness centrality has a primary role, as it is the fraction of shortest paths between node pairs that pass through a node, and thus it is capable of measuring the influence of this node over the information spread through the network [1,17]. Therefore, to detect nodes that interconnect different communities we have to find nodes that are central in terms of betweenness centrality. In general, this search could appear difficult, if no restriction of the domain is done. The idea we present here is based on the consideration that some special nodes exist that exhibit explicitly a role of connectors between two different worlds. To understand this, we have to move towards the perspective of the social internetworking scenario (SIS) [18,3,7,6,8], in which the scope of the action of both people and applications is not confined to a single social network. As a matter of fact, in the current scenario characterized by hundreds of online social networks, a single user can join more of them. This leads to have membership overlap among social networks as expression of different traits of users' personality (sometimes almost different identities), also enabling, as side effect, the passage of information from one social network to another. As a consequence, membership overlap can be viewed as a feature that gives a specific power to users in terms of capability of connecting different worlds. But the good news is that, differently from a generic central node in a social network, a node interconnecting two different social networks (that we call *i-bridge*) often exhibits some explicit information showing this feature, thus it can be easily (automatically) identified. Indeed, often, a user shows in the home page of her account in a social network the link to her account in another social network. As a consequence, it is possible to define crawling strategies that privilege these nodes, allowing their easy discovery. One of these strategies is that called Bridge-Driven Search (BDS) presented in [9].

On the basis of this reasoning, our idea is to use BDS to find a number of seeds for our team formation, possibly iterating the process until a suitable stage is reached. But, the basic question is now: Are all i-bridges good for our purpose? How to compare different i-bridges in terms of capability of generating diversity? On the basis of the above considerations, one could think that the classic measure of betweennes centrality could be used to do this. In this paper, we show that the above hypothesis is not correct. In particular, we show that we need a measure of centrality that, differently from classic betweenness centrality, is able to take into account paths crossing distinct social networks in a different way from internal paths. Thus, we study a new measure of betweenness centrality, called *cross betweenness centrality (CBC)*, which allows us to characterize nodes in a social internetworking scenario in terms of importance w.r.t. inter-social-network information flows and to characterize candidate nodes in terms of suitability to be actors in global team formation.

It is clear that all the above reasoning appears well-founded only if the basic (even intuitive) assertion really holds that i-bridges play the role of ties connecting two social networks in a weak way. Thus, in this paper, as a first contribution, we prove the above claim through an experimental validation.

The plan of this paper is as follows. In the next section, we model our reference scenario. A description of the testbed adopted for the experimental campaign is described in Section 3. In Section 4, we prove the basic claim that i-bridges play the role of ties connecting two social networks in a weak way. In Section 5, we introduce the new measure, called *cross betweenness centrality (CBC)*, needed for our application to detect good i-bridges. An experimental proof about the need and the validity of CBC is presented in Section 6. In Section 7, a review of the related literature is discussed. Finally, in Section 8, we draw our conclusions and identify how to continue our research.

2 The Multi-Social-Network Model

In this section, we model our reference scenario, which is called *social internetworking scenario* (SIS) and takes into account the multiplicity of social networks and their interconnections through `me` edges. A *SIS* is a directed graph $G = \langle N, E\rangle$, where N is the set of *nodes* representing (social network) user accounts and E is the set of *edges* (i.e., ordered pairs of nodes) representing relationships between user accounts. Given a node $a \in N$, we denote by $S(a)$ the social network which a belongs to. E is partitioned into two subsets E_f and E_m. E_f is said the set of *friendship edges* and E_m is the set of `me` *edges*. E_f is such that for each $(a, b) \in E_f$, $S(a) = S(b)$, while E_m is such that for each $(a, b) \in E_m$, $S(a) \neq S(b)$. Each node a of G is associated with the (social network) account of a user joining the social network $S(a)$. An edge $(a, b) \in E_f$ means that the user having the account b is a friend of the user having the account a (both a and b belong to the same social network). An edge $(c, c') \in E_m$ means that c and c' are accounts of the same user in two different social networks. As a consequence, c is an i-bridge and a `me` edge interconnects the two social networks $S(c)$ and $S(c')$. Given a SIS $G = \langle N, E\rangle$, we call *the social networks of* G the set of social networks $\mathcal{S}$ such that for each $Q \in \mathcal{S}$ there exists $n \in N$ such that $S(n) = Q$. Observe that, the graph of a SIS differs from those underlying a single social network because of the presence of `me` edges, which connect nodes of different social networks. From now on, consider given a SIS $G = \langle N, E\rangle$.

3 Tools and Data for Our Experiments

Our experiments have been carried out on real-life data obtained by crawling online social networks. As for the crawling strategy, we used the well-known *BFS* [22], which performs a *Breadth First Search* on a local neighborhood of a seed it starts from. The crawling task was performed by means of the system *SNAKE* [5], which is able to extract not only connections among the accounts of different users in the same social network, but also connections among the accounts of the same user in different social networks. These connections are represented by two standards encoding human relationships, namely XFN (XHTML Friends Network) and FOAF (Friend-Of-A-Friend). This way, we got a dataset consisting of five social networks, namely Flickr, LiveJournal, Google+, MySpace, and Twitter. They are compliant with the XFN or FOAF standards and have been largely analyzed in social network analysis in the past. Starting from this real-life dataset, we extracted several subgraphs for our tests. Each subgraph was obtained by randomly choosing an i-bridge node b and selecting all the nodes having a minimum distance from b less than or equal to 4 (observe that, due to the small diameter of real-life social networks, the chosen distance is significative). Figure 1 shows one of the subgraphs of the real-life dataset, in which black and white nodes stand for users belonging to Flickr and LiveJournal, respectively.

4 Are `me` edges Weak ties?

One of the most fascinating results of social network analysis is due to one of the fathers of this discipline and regards the concept of weak ties [14]. Although a complete notion of the strength of a tie can be given only if we consider dynamic and behavioral information, even the sole structural knowledge about a social network allows us to identify those ties that, informally speaking, connect two dense components keeping a low connection degree between them. The theory presented in [14] and confirmed by years of study on social networks, gives a strong importance to such ties, as they connect different communities, so they can be a formidable vehicle of cross contamination between them. The claim underlying our crowdsourcing application is that i-bridges are good candidates to be actors in a global team formation. But to prove this claim, the first step is to face this issue: Are `me` edges weak ties, in general?

In this section, we try to give an answer to the above question by conducting a suitable experiment on real-life social networks. To detect weak ties, we adopt the strategy proposed in [10]. In particular, we consider an edge e between the nodes n_1 and n_2 and we check if the removal of this edge would increase the distance between n_1 and n_2 to a value strictly more than two. If this occurs, then e can be considered as a weak tie. This experiment is carried out on the whole dataset described in Section 3 consisting of a set of $171,982$ normal edges (i.e., non-`me` edges) and a set M of 79 `me` edges. At the end of the experiment, the approach described above detected a set W of 5619 weak ties among the set E of $172,061$

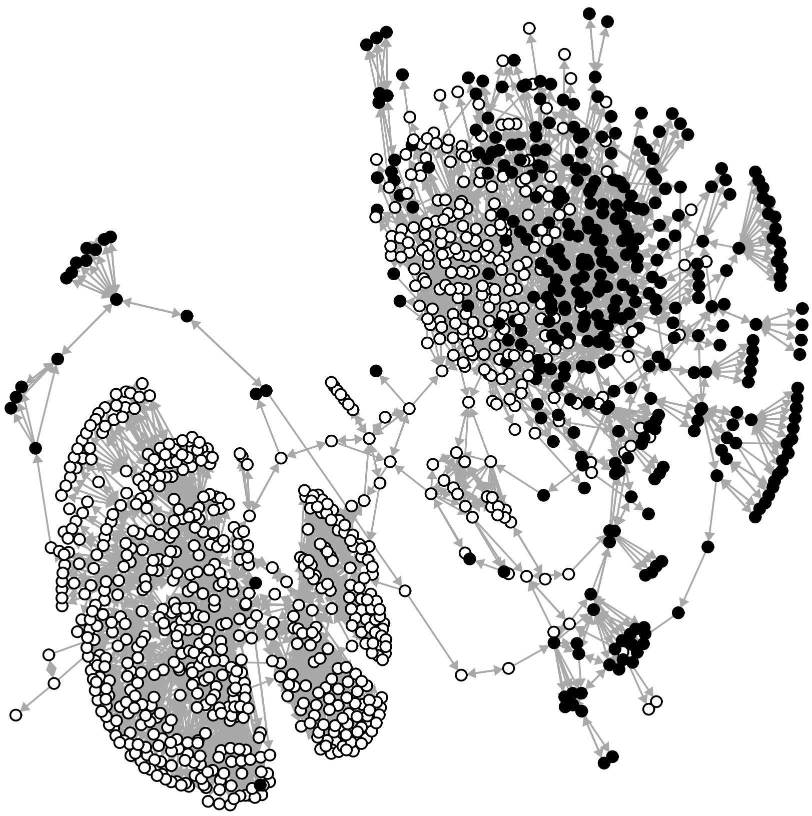


Fig. 1. Visualization of one subgraph

edges and 60 of them were also `me` edges. We calculated the percentage of `me` edges which are weak ties as $\frac{|M \cap W|}{|M|} = 0.76$, the percentage of `me` edges $\frac{|M|}{|E|} = 4.6 \cdot 10^{-4}$ and the percentage of weak ties $\frac{|W|}{|E|} = 3.3 \cdot 10^{-2}$. From the above results, we conclude that the probability that a `me` edge is a weak tie is high, whereas the probability that a generic edge is a weak tie is very low. Because $|M|$ and $|W|$ are much lesser than $|E|$, the result that $\frac{|M \cap W|}{|M|} = 0.76$ demonstrates that a strong relation between weak ties and `me` edges exists. Thus, the experiment concludes that the correlation between weak ties and `me` edges exists, so that we can sight potential powerful roles for such edges in a social internetworking scenario. It is worth remarking that the interpretation of `me` edges as weak ties means that, given a user u, her account in a social network S sees her account in another social network T (in case a `me` edge from S to T is established by u) as a weak tie. This means that u can be used as powerful disseminator of information across different communities, each belonging to a different social network. Obviously, the more the number of `me` edges of a user u, the higher her strength (in the

Granovetter sense [14]) in the network. Observe that the above conclusion is not in contradiction with the results given in [8], where the presence of `me` edges into user accounts has been proven to be assortative. Indeed, one could think that, if the friends of a user u assortatively declare `me` edges from the social network S to the social network T, as done by u, we obtain a very dense clique of users invalidating the result about the correspondence between `me` edges and weak ties. However, [8] shows that the *strict* (i.e., towards the same social network T) assortativity does not hold for the most representative real-life social networks (i.e., Facebook). In other words, the friends of u assortatively declare `me` edges from S to any other social network. Thus, the contradiction does not exist.

5 Measuring the Suitability of Diversity Generators: Cross Betweenness Centrality

After having proved, in Section 4, that `me` edges are weak ties, we know that our application is well-founded in the sense that all the i-bridges that we are able to find by using a crawling strategy as BDS [9] are good candidates to play the role of diversity generators in team formation. In this section we face a second important issue: Are all the candidates the same in terms of suitability to our application? In general, betweenness centrality (BC) [11] is used to detect weak ties (and also their *structural* strength). Unfortunately, as we will show in Section 6, the above claim cannot be applied to the case of i-bridges, in the sense that it is not able to measure their structural strength in terms of connectors of two social networks. Therefore, in this section, we introduce a new measure called *cross betweenness centrality (CBC)* to *rate* candidates in our application, overcoming the limits of BC. The need and the validity of CBC is shown in the next section. Recall that we are interested in a measure able to take into account paths crossing distinct social networks in a different way from internal paths. Even though the definition of betweenness centrality does not explicitly take into account the presence of the multiplicity of social networks, it could happen that the real-life structure of the interconnections among distinct social networks (i.e., i-bridges) is such that BC automatically favors nodes belonging to the frontier of each social network, as paths are in some way forced to cross them. Intuitively, the above claim is true if the density of the involved social networks is comparable. Otherwise, we expect that the most dense social network works as an accumulation point, biasing the centrality towards it. However, also in this case, the role of i-bridges is still crucial, so we would like not to miss it.

The definition of *cross betweenness centrality (CBC)* is the following. Let $\Omega \subseteq \mathcal{S}$. Given a node $n \in N$, we denote the *cross betweenness centrality* of n w.r.t. Ω as:

$$CBC(n, \Omega) = \begin{cases} \sum_{s,t \in N, s \neq n, t \neq n, S(s) \neq S(t), S(t) \in \Omega} \frac{\sigma_{st}(n)}{\sigma_{st}} & \textit{if } \sigma_{st} > 0 \\ 0 & \textit{otherwise} \end{cases}$$

where σ_{st} is the total number of the shortest paths from s to t and $\sigma_{st}(n)$ is the number of those shortest paths from s to t passing through n. In this definition,

Table 1. Results obtained for the first subgraph

Type of node	*BC*	*CBC*
Bridges	1,242,081.10	42.67
Power Users	1,543,513.59	4.43
Normal Users	3,795.34	0.01

Ω is a subset of the social networks of the SIS (see Section 2) and allows the computation of the cross betweenness centrality of a node to be limited (if this is desired) to a subset of the social networks of the SIS. In the definition of CBC, considered paths are only those (*i*) linking two nodes belonging to different social networks and (*ii*) having the target node (t) belonging to one social network in Ω (it does not matter whether the source node s belongs to a social network in Ω). In particular, we compute how many times the node n is involved in this kind of path. Interestingly, if n belongs to a fragment of a social network not connected with the rest of the SIS, then $CBC(n, \Omega) = 0$.

Observe that the following relation between cross betweenness centrality and the classical betweenness centrality can be proved: $BC(n) = CBC(n, \Omega) + CBC(n, \overline{\Omega}) + IBC(n)$, where $IBC(n) = \sum_{s,t \in N, s \neq n, t \neq n, S(s)=S(t)} \frac{\sigma_{st}(n)}{\sigma_{st}}$ and $\overline{\Omega} = \mathcal{S} \setminus \Omega$. A direct consequence of this results is that, in the trivial case of a single-social-network SIS, $BC(n) = IBC(n)$. Indeed, no inter-social-network contribution occurs.

6 Need and Validity of CBC: An Experimental Proof

In this experimental section, we show that BC is not able to capture the capability of i-bridges to be central w.r.t. cross-social-network paths. Then, we show that CBC is a measure that can be used in our application to compare different i-bridges in terms of suitability to be actors in global team formation. For this purpose, we partitioned nodes into three categories:

1. i-bridges, which are nodes with a `me` edge;
2. power users, which are non-i-bridge nodes whose degree is equal to, or higher than, the average degree of all nodes;
3. normal users, which are neither i-bridges nor power users.

Then, we computed the average values of BC and CBC for each category. The experiments presented in this section are carried out on two of the subgraphs described in Section 3. The results obtained for the first and second subgraphs are reported in Tables 1 and 2.

From the analysis of these tables, we note that there is no correlation between BC and node cathegory. Indeed, in Table 1, i-bridges and power users have comparable values, whereas normal users have a value that is about 3 magnitude orders lesser than the previous ones. By contrast, in Table 2, power users and

Table 2. Results obtained for the second subgraph

Type of node	*BC*	*CBC*
Bridges	105.01	43.33
Power Users	1,480,655.07	5.14
Normal Users	1,092,715.52	0.04

normal users have comparable values, whereas i-bridges have a value 4 magnitude orders lesser than the previous ones. Thus, it is evident that betweenness centrality is not able to correctly identify i-bridge nodes.

Consider now cross betweenness centrality. By looking at Tables 1 and 2, we observe that, for each node category, there is a great uniformity in the corresponding values. Even more interesting, i-bridges have a value of CBC always higher than power users (about one magnitude order), which, in turn, show a value much higher than normal users (about two magnitude orders). Therefore, the distinction among the three categories of nodes is evident by taking cross betweenness centrality into account. Thus, even our expectation about cross betweenness centrality is confirmed by analyzing real-life social networks.

7 Related Work

The concept of centrality, as applied to the context of human communication, was first introduced by Bavelas in 1948 [2]. He mainly focused on communication in small groups and he hypothesized a relationship between structural centrality and influence in group processes. More recently, Leavitt [15], Shaw [19] and Goldberg [13] proposed studies on speed, activity and efficiency in solving problems, on personal satisfaction and on leadership in small group settings. The concept of centrality is motivated by the idea that a person who is close to others can have access to more information, a higher status, more power, a greater prestige, or a greater influence [12] than others. Indeed, this person can facilitate or inhibit the communication of others and is, therefore, in a position to mediate their information access, power, prestige, or influence. Among all the centrality measures, betweenness centrality is one of the most popular, and its computation is the core component of a range of algorithms and applications. Both Bavelas [2] and Shaw [19] suggested that, when a person is strategically located in the middle of communication paths linking other users, she is central. A person in such a position can influence the group by holding or distorting information. By the way, the development of betweenness centrality is generally attributed to the sociologist Linton Freeman [11]. Over the past few years, betweenness centrality has become a popular strategy to measure node influence in complex networks, such as social networks. For this purpose, a lot of new metrics based on betweenness centrality have been already defined [17]. A concept strongly related to edge importance is edge classification. This task is usually performed on the basis of the kind (and, hence, of the "strength") of the relationship the edge represents. Under this assumption, an edge could be a strong or a weak tie. The concept of tie strength was introduced by Mark Granovetter in his very popular paper entitled "The Strength of Weak Ties" [14]. He identified four main features contributing to outline the strength of a tie, namely: amount of time, intimacy, intensity and reciprocal services. Finally, a first characterization of the nodes of a SIS has been proposed in [4]. However, no experimental validation has been provided therein.

8 Conclusion and Future Work

In this paper, we have presented a preliminary idea of a crowdsourcing application aimed at driving the process of global team formation to obtain diversity in the team. This paper presents a first step towards the concrete definition of the above application consisting in the identification of an effective measure that can be used to select seed nodes in the team formation. A first preliminary experimental validation has been provided showing that our idea is well-founded. The next steps are to further validate the new measure and to design the social web application in detail. We plan to do this in our future research.

Acknowledgments. This work has been partially supported by the TENACE PRIN Project (n. 20103P34XC) funded by the Italian Ministry of Education, University and Research and by the Program "Programma Operativo Nazionale Ricerca e Competitività" 2007-2013, Distretto Tecnologico CyberSecurity funded by the Italian Ministry of Education, University and Research.

References

1. Barthelemy, M.: Betweenness centrality in large complex networks. The European Physical Journal B-Condensed Matter and Complex Systems 38(2), 163–168 (2004)
2. Bavelas, A.: A Mathematical Model for Small Group Structures. Human Organization 7(3), 16–30 (1948)
3. Buccafurri, F., Foti, V., Lax, G., Nocera, A., Ursino, D.: Bridge Analysis in a Social Internetworking Scenario. Information Sciences 224, 1–18 (2013)
4. Buccafurri, F., Lax, G., Nicolazzo, S., Nocera, A., Ursino, D.: Measuring Betweenness Centrality in Social Internetworking Scenarios. In: Demey, Y.T., Panetto, H. (eds.) OTM 2013 Workshops 2013. LNCS, vol. 8186, pp. 666–673. Springer, Heidelberg (2013)
5. Buccafurri, F., Lax, G., Nocera, A., Ursino, D.: A system for extracting structural information from Social Network accounts. Technical Report. Available from the authors
6. Buccafurri, F., Lax, G., Nocera, A., Ursino, D.: Crawling Social Internetworking Systems. In: Proc. of the International Conference on Advances in Social Analysis and Mining (ASONAM 2012), Istanbul, Turkey, pp. 505–509. IEEE Computer Society (2012)
7. Buccafurri, F., Lax, G., Nocera, A., Ursino, D.: Discovering Links among Social Networks. In: Flach, P.A., De Bie, T., Cristianini, N. (eds.) ECML PKDD 2012, Part II. LNCS, vol. 7524, pp. 467–482. Springer, Heidelberg (2012)

8. Buccafurri, F., Lax, G., Nocera, A., Ursino, D.: Internetworking assortativity in Facebook. In: Proc. of the International Conference on Social Computing and its Applications (SCA 2013), Karlsruhe, Germany, pp. 335–341. IEEE Computer Society (2013)
9. Buccafurri, F., Lax, G., Nocera, A., Ursino, D.: Moving from social networks to social internetworking scenarios: The crawling perspective. Information Sciences 256, 126–137 (2014)
10. Easley, D., Kleinberg, J.: Networks, crowds, and markets, vol. 8. Cambridge University Press, Cambridge (2010)
11. Freeman, L.C.: Centrality in Social Networks Conceptual and Clarification. Social Networks 1(3), 215–239 (1979)
12. Friedkin, N.E.: Theoretical foundations for centrality measures. American Journal of Sociology 96(6), 1478–1504 (1991)
13. Goldberg, S.C.: Influence and leadership as a function of group structure. Journal of Abnormal and Social Psychology 51(1), 119–122 (1955)
14. Granovetter, M.: The strength of weak ties. American Journal of Sociology 78(6), 1360–1380 (1973)
15. Leavitt, H.J.: Some effects of a certain communication patterns on group performance. Journal of Abnormal and Social Psychology 46(1), 38–50 (1951)
16. Lim, S., Ncube, C.: Social networks and crowdsourcing for stakeholder analysis in system of systems projects. In: Proc. of the International Conference on System of Systems Engineering (SoSE 2013), Maui, Hawaii, USA, pp. 13–18. IEEE (2013)
17. Newman, M.: A measure of betweenness centrality based on random walks. Social Networks 27(1), 39–54 (2005)
18. Okada, Y., Masui, K., Kadobayashi, Y.: Proposal of Social Internetworking. In: Shimojo, S., Ichii, S., Ling, T.-W., Song, K.-H. (eds.) HSI 2005. LNCS, vol. 3597, pp. 114–124. Springer, Heidelberg (2005)
19. Shaw, M.E.: Group structure and the behavior of individuals in small groups. Journal of Psychology 38(1), 139–149 (1954)
20. Surowiecki, J., Silverman, M.: The Wisdom of Crowds. American Journal of Physics 75(2), 190–192 (2007)
21. Vukovic, M.: Crowdsourcing for enterprises. In: Proc. of the International Conference on Services-I 2009, Los Angeles, CA, USA, 2009, pp. 686–692. IEEE Computer Society Press (2009)
22. Ye, S., Lang, J., Wu, F.: Crawling online social graphs. In: Proc. of the International Asia-Pacific Web Conference (APWeb 2010), Busan, Korea, pp. 236–242. IEEE (2010)

Effectiveness of Incorporating Follow Relation into Searching for Twitter Users to Follow

Tomoya Noro and Takehiro Tokuda

Department of Computer Science, Tokyo Institute of Technology
Meguro, Tokyo 152-8552, Japan
{noro,tokuda}@tt.cs.titech.ac.jp

Abstract. Twitter is one of the most popular microblogging services that facilitate real-time information collection, provision, and sharing. Following influential Twitter users is one way to get valuable information related to a topic of interest efficiently. Recently many researches on this issue have been done and, in general, it is said that the follow relation is not useful for measuring user influence. In this paper, we study effectiveness of incorporating not only the tweet activity (retweet and mention) but also the follow relation into searching for good Twitter users to follow for getting information on a topic of interest. We present a method for finding Twitter users based on both the follow relation and the tweet activity, and show the follow relation could improve the performance as compared with methods based on only the tweet activity.

Keywords: Social network analysis, microblog, Twitter, search, influential users, power iteration algorithm.

1 Introduction

Currently Twitter has become a more and more important platform of information collection, provision, and sharing. If we would like to get information of a topic of interest and discuss the topic with others on a daily basis, we usually follow some users who provide valuable information on the topic [1]. For example, if we look for information about dementia, we could find that some doctors and care staff members deliver information about the topic, and some people who have family members with dementia post tweets about their daily care. We can get various information on dementia by following such users. However, it is not easy for us to find good users to follow in a massive number of users.

Many researches on this issue have been done recently. Measuring user influence on a particular topic will be one solution. They measure each user's influence based on the tweet content, the follow relation, the tweet activity such as retweet and mention, and so on, and some of them pointed out that the follow relation is not useful for measuring user influence. Cha et al. investigated

[1] In this paper, we do not consider temporary topics such as incidents and events (natural disaster, terrorism, FIFA World Cup, etc). If we would like to get information on such topics, we would take different actions such as keyword search.

characteristics of Twitter users, then concluded users who have many followers are popular but not necessarily influential, while users who are retweeted or mentioned many times have ability to post valuable tweets or ability to engage others in conversation [2]. We have also been working on this issue and showed a user search method based on the tweet activity outperforms a method based on the follow relation [6].

Here is one question. Is the follow relation really useless for finding good users to follow? Actually some users follow almost all of their followers. We can easily get many followers if we search for such users, follow them, wait for them to follow us, and remove them if they do not follow us. The follow relation built in this way is meaningless since most of the followers may not be interested in our tweets and we have little influence on them. However, in general, influential users have many followers. Users who have a small number of followers may not be good users to follow since they should have more followers if they provide a lot of valuable information. The search accuracy could be improved by dealing with both the tweet activity and the follow relation.

In this paper, we study effectiveness of incorporating not only the tweet activity but also the follow relation into searching for good Twitter users to follow for getting information on a topic of interest. We present a method for finding good users to follow based on both the follow relation among users and the tweet activity of each user. In evaluation, we compare the method with other methods without taking the follow relation into account and show incorporating both the tweet activity and the follow relation could improve the search performance.

This paper is organized as follows. In section 2, we discuss some researches on searching Twitter users to follow on a topic of interest. We present our method based on both the follow relation and the tweet activity in section 3, then show some evaluation results in section 4. Lastly we conclude this paper in section 5.

2 Related Work

Twitter provides its own services for user search and recommendation. Given some keywords, Twitter mainly shows some users whose screen names or profiles match the keywords and does not care whether they actually post tweets related to the keywords. The Twitter recommendation service shows users based on the follow relation (users who have mutual followers and/or friends), and does not consider their activity and the tweet content either.

Twittomender [3] finds users related to a particular user or query by using lists of followers, friends and terms in the user's tweets. TwitterRank [8] considers the follow relation and topical similarity to find influential users. Both of the methods are based on the follow relation and they do not consider the tweet activity such as retweet and mention.

Leavitt et al. [5] measured user influence by using ratio of being retweeted and mentioned to the number of tweets the user posted. They do not consider the follow relation. Anger et al. [1] defined user influence based on ratio of retweeted tweets and ratio of the user's followers who retweeted the user's tweets or mentioned the user. Although they consider both the tweet activity and the follow relation, they use the follow relation to observe how many followers retweeted the user's tweets or mentioned the user and do not consider who follows whom.

We presented a method for finding good users to follow for getting information about a topic of interest by using the tweet activity [6]. Although we showed that a search method based on the tweet activity outperforms a method based on the follow relation, we study effectiveness of considering both the tweet activity and the follow relation in this paper.

3 Method for Finding Good Twitter Users to Follow

3.1 Overview

Our process of finding good users to follow on a topic of interest goes as follows.

1. Given some keywords representing the topic of interest, collect tweet data and user data by using the Twitter APIs.
2. Create a user reference graph based on the follow relation and a user-tweet reference graph based on both the tweet activity and the follow relation.
3. Calculate score of each user from the two graphs and rank the users.

The data collection process in the first step goes as follows.

1. Given some keywords representing a topic of interest, get tweets matching the keywords posted in the last N days. Duplicate tweets (exactly the same tweet text posted by the same user) are removed to exclude spammers who post the same tweets repeatedly. Let this tweet set be T_0.
2. For each tweet in T_0, get ID and poster's name of the tweet and user names in the tweet text (user mention). If the tweet is a reply tweet/retweet, get ID and poster's name of the replied/retweeted tweet. Let the set of tweets and the set of users be T_{all} and U_{all} respectively.
3. Get the follow relation among users in U_{all} ($F \subseteq U_{all} \times U_{all}$).

3.2 Score Calculation

In order to define the score of each user, we assume the followings.

1. Users who post many valuable tweets about the topic are worth following.
2. Valuable tweets attract attention from many users.
3. Each user pay attention to tweets the user retweets or replies to.
4. Each user also pay attention to tweets posted by the user's friends.

The first assumption means that users who post many tweets related to the topic should be ranked higher. However, some users who post many valueless tweets such as spam tweets will also be ranked higher if we consider only this assumption. To exclude such users, we take the other assumptions into account. A user's retweeting or replying to a tweet means that the user is interested in the tweet. Each user may pay attention to a tweet posted by the user's friends to some extent even if the user did not retweet or reply to the tweet.

Based on these assumptions, we define the score of each user u as follows.

$$\begin{aligned}\mathrm{Score}(u) &= \mathrm{TC}(u)^{w_c} \times \mathrm{UI}(u)^{w_i} \times \mathrm{FR}(u)^{w_f} \\ &\quad \text{such that} \quad w_c + w_i + w_f = 1 \wedge w_c \geq 0 \wedge w_i \geq 0 \wedge w_f \geq 0 \end{aligned} \quad (1)$$

$\mathrm{TC}(u)$, $\mathrm{UI}(u)$, and $\mathrm{FR}(u)$ are respectively "tweet count (TC) score", "user influence (UI) score", and "follow relation (FR) score" of user u ranging between 0 and 1. The TC score is based on the number of tweets each user posted, and reflects the first assumption. The FR score is based on the follow relation among users, and reflects the second and forth assumptions. The UI score is based on both the tweet activity and the follow relation, and reflects the second, third and forth assumptions.

3.3 Tweet Count Score (TC Score)

The TC score is calculated by counting not only each user's original tweets but also retweets in T_0. We count retweets as each user's own tweets [2]. The score is normalized so that the largest value should be 1.

$$\mathrm{TC}(u) = \frac{\log(1 + |\{t | t \in T_0 \wedge t.user.id = u.id\}|)}{\max_{u' \in U_{all}} \log(1 + |\{t | t \in T_0 \wedge t.user.id = u'.id\}|)} \quad (2)$$

$t.user.id$ indicates poster's ID of tweet t and $u.id$ indicates ID of user u.

3.4 User Influence Score (UI Score)

The basic idea is as follows.

1. If user u_i retweets or replies to user u_j's tweet, u_j has an influence on u_i.
2. Users who post many tweets paid attention to by many users are influential, especially if their tweets are often paid attention to by influential users.

How much each tweet is paid attention to by others is measured according to the tweet activity (retweet and reply) and the follow relation. Based on this idea, we define not only the UI score of each user but also tweet influence score (TI score) of each tweet. The UI score is calculated using the TI score of tweets and retweets posted by the user, and the TI score is calculated using the UI score of users who pay attention to the tweet.

We create a user-tweet reference graph consisting of user nodes (U_{all}), tweet nodes (T_{all}), and directed edges each of which connects a user node and a tweet

[2] The retweet activity is incorporated into both the TC score and the UI score. The number of times each user retweeted is considered in the TC score while the user-tweet relation (who retweeted what) is considered in the UI score.

node. The reference graph is represented as combination of three adjacency matrices A_t, A_r, and A_s.

$$A_t(t_i, u_j) = \begin{cases} 1 & \text{if } t_i \text{ is tweeted/retweeted by } u_j \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$A_r(u_j, t_i) = \begin{cases} 1 & \text{if } u_j \text{ retweets/replies to } t_i \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$A_s(u_j, t_i) = \begin{cases} 1 & \text{if } u_j \text{ follows at least 1 user who tweets/retweets } t_i \\ \alpha & \text{otherwise } (0 < \alpha \leq 1) \end{cases} \tag{5}$$

t_i and u_j indicates the i-th tweet and the j-th user respectively ($1 \leq i \leq |T_{all}|$ and $1 \leq j \leq |U_{all}|$). A_t and A_r are derived from the tweet activity of each user, and A_s is derived from the follow relation among users. A_t represents what (tweet) is tweeted or retweeted by whom (user), and A_r and A_s respectively represent who retweets or replies to what and who sees what. The follow relation will be ignored if α is equal to 1.

These adjacency matrices are transformed into the following two matrices.

$$B_t(t_i, u_j) = \frac{A_t(t_i, u_j)}{\sum_k A_t(t_i, u_k)} \tag{6}$$

$$B_a(u_j, t_i) = \begin{cases} \frac{A_r(u_j,t_i)}{\sum_k A_r(u_j,t_k)}(1-d) + \frac{A_s(u_j,t_i)}{\sum_k A_s(u_j,t_k)}d & \text{if } \sum_k A_r(u_j, t_k) \neq 0 \\ \frac{A_s(u_j,t_i)}{\sum_k A_s(u_j,t_k)} & \text{otherwise} \end{cases} \tag{7}$$

d is a damping factor of $0 < d < 1$. Transformation of A_r and A_s into B_a reflects the third and forth assumptions of good users to follow described in section 3.1. Each user pay attention to tweets the user retweets or replies to, and the user also watches all tweets at a certain rate of d regardless of the user's activity of retweet and reply. Tweets posted or retweeted by the user's friends are more likely to be seen than the other tweets, and the idea is also included.

The UI score and the TI score are calculated as follows.

$$\mathbf{u} = B_t^T \mathbf{t} \qquad\qquad \mathbf{t} = B_a^T \mathbf{u} \tag{8}$$

u and **t** indicate a column vector of the UI score of all users and a column vector of the TI score of all tweets respectively. We can calculate the UI score and the TI score using the power iteration method. Lastly the UI score of each user is normalized so that the largest value should be 1.

$$\mathrm{UI}(u_j) = \frac{\mathbf{u}(j)}{\max_k \mathbf{u}(k)} \tag{9}$$

3.5 Follow Relation Score (FR Score)

The FR score is calculated based on the follow relation using PageRank [7]. A user reference graph is created from the follow relation F. Adjacency matrix of the graph is represented as follows.

$$A_f(u_i, u_j) = \begin{cases} 1 & \text{if } u_i \text{ follows } u_j \text{ i.e. } (u_i, u_j) \in F \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

$$B_f(u_i, u_j) = \begin{cases} \frac{A_f(u_i,u_j)}{\sum_k A_f(u_i,u_k)}(1-d) + \frac{d}{|U_{all}|} & \text{if } \sum_k A_f(u_i, u_k) \neq 0 \\ \frac{1}{|U_{all}|} & \text{otherwise} \end{cases} \tag{11}$$

$$\mathbf{f} = B_f^T \mathbf{f} \tag{12}$$

u_i and u_j indicates the i-th user and the j-th user respectively, and d is a damping factor. $\mathbf{f}$ indicates the column vector of the FR score of all users.

Unlike normalization of the TC score and the UI score, the FR score of each user is not divided by the maximum value. As described in section 1, users who have many followers are not necessarily influential, and it is said that the follow relation is not useful for measuring user influence. However, we think the follow relation could be used for excluding uninfluential users who have few (influential) followers. Instead of dividing the FR score of each user by the maximum value, we set upper limit of the FR score to the minimum score of the top-$P\%$ users and divide the score of each user by the limit.

$$\mathrm{FR}(u_i) = \frac{\min(\mathbf{f}(i), \mathit{limit})}{\mathit{limit}} \tag{13}$$

limit indicates the minimum FR score of the top-$P\%$ users. This normalization can weaken influence of the users who have high FR score since the score of all of the top-$P\%$ users will be set to 1.

Some alternative ways for normalization may be considered. For example, some may think of normalization by setting the upper limit to a predetermined proportion to the maximum value (e.g. $\mathit{limit} = 0.1 \times \max_k \mathbf{f}(k)$) or determining the number of users to be capped (e.g. Top-50 users). However, the number of users to be ranked depends on topics of interest, and distribution of the FR score also varies by the topics. We think that determining the percentage of users to be capped is better from our observation.

4 Evaluation

4.1 Experimental Setup

We selected the following 7 Japanese keywords (in Japanese characters) as input query representing topics of interest: "nuclear power", "animal test", "whaling", "dementia", "digital book", "basic income" and "fair trade". We chose these topics since we expect that tweets related to the topics are posted on a daily basis (independent of season). Tweets and other data were collected 6 times on different days. For each time, we get tweets posted in the last 5 days. The average number of tweets and users we collected is shown in Table 1. "Reply" means tweets replying to tweets specified in "reply-to" attribute, and "Mention" means tweets including user names but not specifying their target tweets.

Table 1. The average number of tweets and users

Keyword	$\lvert T_0 \rvert$ Total	Retweet	Reply	Mention	$\lvert T_{all} \rvert$	$\lvert U_{all} \rvert$
nuclear power	26,937.7	14,878.0	1,008.7	1,336.0	28,124.0	13,435.3
animal test	2,591.7	1,539.8	185.7	126.5	2,818.3	1,349.3
whaling	4,057.7	1,045.7	254.0	321.0	4,249.7	3,112.7
dementia	5,497.5	1,670.7	832.3	163.5	6,255.0	4,959.3
digital book	19,208.0	4,408.2	1,307.8	1,273.3	20,333.5	12,976.8
basic income	400.7	148.8	68.3	19.8	449.0	251.5
fair trade	779.2	364.8	70.7	14.7	849.8	662.2

Table 2. Value of each parameter

d in Eqs. (7) and (11)	0.15
P for FR(+lim)	5%
α in Eq. (5) for UI(+fol)	0.1
w_c and w_i in Eq. (1) for TC+UI	0.6 and 0.4
w_c, w_i and w_f in Eq. (1) for TC+UI+FR	0.6, 0.2, and 0.2

We set up the following methods for comparison.

TC+UI(+fol)+FR(+lim): Rank users based on the TC score, the UI score with the follow relation, and the FR score with upper limit.

TC+UI(-fol)+FR(+lim): Rank users based on the TC score, the UI score without the follow relation, and the FR score with upper limit (α in Eq. (5) is set to 1).

TC+UI(+fol)+FR(-lim): Rank users based on the TC score, the UI score with the follow relation, and the FR score without upper limit (normalization of the FR score is done by dividing each score by the maximum value).

TC+UI(-fol)+FR(-lim): Rank users based on the TC score, the UI score without the follow relation, and the FR score without upper limit.

TC+UI(+fol): Rank users based on the TC score and the UI score with the follow relation (w_f in Eq. (1) is set to 0).

TC+UI(-fol): Rank users based on the TC score and the UI score without the follow relation

TC: Rank users based on only the TC score.

UI(+fol): Rank users based on only the UI score with the follow relation.

UI(-fol): Rank users based on only the UI score without the follow relation

FR(-lim): Rank users based on only the FR score without upper limit

We carried out a preliminary experiment using tweet data collected on different days to determine the parameters appeared in section 3. The value of each parameter is shown in Table 2.

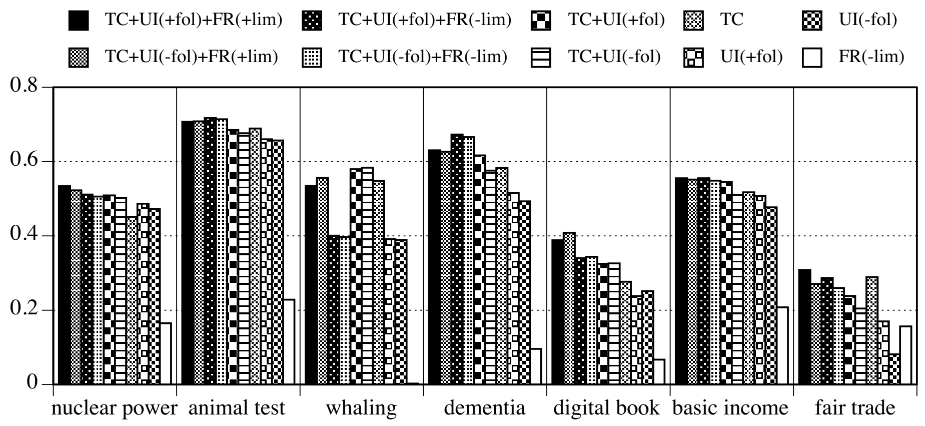


Fig. 1. Average nDCG of each method

Evaluation is done on the top 20 users ranked by each method with respect to normalized discounted cumulative gain (nDCG) [4].

$$\mathrm{DCG}_{20} = rel_1 + \sum_{i=2}^{20} \frac{rel_i}{\log_2 i} \qquad \mathrm{maxDCG}_{20} = 2 + \sum_{i=2}^{20} \frac{2}{\log_2 i} \tag{14}$$

$$\mathrm{nDCG}_{20} = \frac{\mathrm{DCG}_{20}}{\mathrm{maxDCG}_{20}} \tag{15}$$

rel_i indicates relevance score between the user ranked i-th and the input keyword, which is judged on a scale of 0 to 2. Users who often post related tweets are assigned the score of 2, while users who rarely post related tweets are assigned the score of 0. Users who post a lot of unrelated tweets like advertisement and spams are also assigned the score of 0. The judgment was done by watching their tweets posted after the data collection period (for about one month) to check whether they continuously post related tweets.

4.2 Result

The result is shown in Figure 1. We can see that considering the follow relation in calculation of the UI score improves the search result on average (e.g. methods including UI(+fol) vs methods including UI(-fol)).

Except for the case of "whaling", incorporating the FR score is also effective (e.g. TC+UI(+fol)+FR(+lim) vs TC+UI(+fol)). The FR(-lim) method found no relevant user in the case of "whaling". When incidents related to whaling occur, many major news organizations will post tweets related to the topic and will get high ranking in the FR score since they have many followers. However, their interest is not always focused on the topic. On the other hand, users who usually talk about or discuss the topic do not have strong follow relation with others

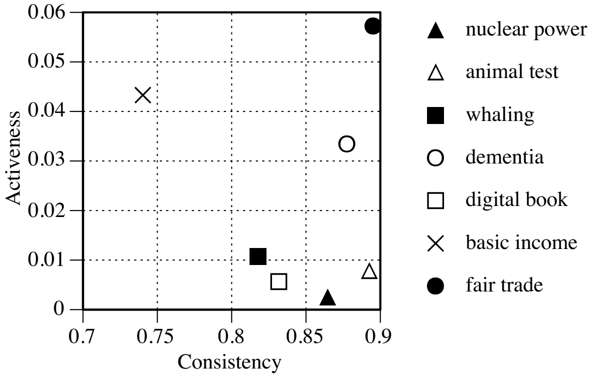


Fig. 2. Consistency and activeness of each keyword

compared to such news organizations. We think this is why the performance were not improved by incorporating the FR score.

Effectiveness of setting the upper limit for normalization of the FR score differs according to keywords (e.g. TC+UI(+fol)+FR(+lim) vs TC+UI(+fol)+FR (-lim)). In the case of "whaling", a large decline caused by incorporating the FR score is prevented. This is because influence of users who is ranked high on the FR score but do not focus on the topic (e.g. major news organizations) are reduced. Setting the upper limit worsen the search performance in the case of "dementia" since, unlike the case of "whaling", some users who usually focus on the topic also have strong follow relation with others and difference between their FR score and the score of major news organizations is not large.

If we compare methods including UI(+fol) with methods including UI(-fol) in more details, we can see that incorporating the follow relation in calculation of the UI score seems not to improve the search performance in the case of "whaling" and "digital book". To analyze this issue, we calculate two measures. One measure is "consistency", how much their tweet activity (retweet and reply) is consistent with their follow relation, and the other measure is "activeness", how many tweets posted by their friends they retweet or reply to. They are defined as follows.

$$\text{Consistency} = \frac{Consistent}{Consistent + Inconsistent} \tag{16}$$

$$\text{Activeness} = \frac{Consistent}{Consistent + Ignored} \tag{17}$$

"*Consistent*" means the number of times they retweeted or replied to their friends' tweets and retweets, and "*Inconsistent*" means the number of times they retweeted or replied to tweets and retweets posted by none of their friends. "*Ignored*" means the number of times they do not retweeted nor replied to their friends' tweets and retweets. From the result shown in Figure 2, we can see both consistency and activeness of "whaling" and "digital book" are low compared with other keywords. We guess users who are interested in the topics are likely to see tweets of non-following users and to communicate with them while they do not communicate with their friends so much. This situation would occur if they usually talk about or discuss topics of interest by using hashtags. It can also be seen in the case of "basic income" but, in this case, many of them communicate with their friends as well as other users (activeness is high in the case of "basic income"). On the other hand, both consistency and activeness of "fair trade" and "dementia" are high. In both cases, we can see effectiveness of incorporating the follow relation into calculation of the UI score. The percentage of retweets, reply tweets and mention tweets of "digital book" and "whaling" is low (36.3% and 40.0% respectively) as shown in Table 1, which would also be one factor that worsens the performance.

5 Conclusion

In this paper, we presented a method for finding good Twitter users to follow for getting information about a topic of interest based on both the tweet activity and the follow relation, and showed incorporating both of them improves the search performance on average except for the case that performance of the method based only the follow relation is extremely bad. We also measured "consistency" and "activeness", and found effectiveness of incorporating the follow relation into the UI score is high if the two scores are high.

References

1. Anger, I., Kittl, C.: Measuring influence on Twitter. In: 11th International Conference on Knowledge Management and Knowledge Technologies, vol. 31. ACM Digital Library (2011)
2. Cha, M., Haddadi, H., Benevenuto, F., Gummadi, K.P.: Measuring user influence in Twitter: The million follower fallacy. In: 4th International AAAI Conference on Weblogs and Social Media, pp. 10–17 (2010)
3. Hannon, J., Bennett, M., Smyth, B.: Recommending Twitter users to follow using content and collaborative filtering approaches. In: 4th ACM Conference on Recommender Systems, pp. 199–206 (2010)
4. Jarvelin, K., Kekalainen, J.: Cumulated gain-based evaluation of IR techniques. ACM Transactions on Information Systems 20(4), 422–446 (2002)
5. Leavitt, A., Burchard, E., Fisher, D., Gilbert, S.: The influentials: New approaches for analyzing influence on Twitter. Web Ecology Project (2009), `http://www.webecologyproject.org/2009/09/analyzing-influence-on-twitter/`
6. Noro, T., Ru, F., Xiao, F., Tokuda, T.: Twitter user rank using keyword search. In: Information Modelling and Knowledge Bases XXIV. Frontiers in Artificial Intelligence and Applications, vol. 251, pp. 31–48. IOS Press (2013)
7. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web. Tech. rep., Stanford University (1998)
8. Weng, J., Lim, E.P., Jiang, J., He, Q.: TwitterRank: Finding topic-sensitive influential Twitterers. In: 3rd ACM International Conference on Web Search and Data Mining. pp. 261–270 (2010)

Improving the Scalability of Web Applications with Runtime Transformations

Esteban Robles Luna[1], José Matías Rivero[1,2], Matias Urbieta[1,2], and Jordi Cabot[3]

[1] LIFIA, Facultad de Informática, UNLP, La Plata, Argentina
{esteban.robles,mrivero,matias.urbieta}@lifia.info.unlp.edu.ar
[2] Also at Conicet
[3] École des Mines de Nantes / INRIA
jordi.cabot@inria.fr

Abstract. The scalability of modern Web applications has become a key aspect for any business in order to support thousands of concurrent users while reducing its computational costs. If a Web application does not scale, a few hundred users can take the application down and as a consequence cause business problems in their companies. In addition, being able to scale a Web application is not an easy task, as it involves many technical aspects such as architecture design, performance, monitoring and availability that are completely ignored by current Model Driven Web Engineering approaches. In this paper we present a model-based approach that uses runtime transformations for overcoming scalability problems in the applications derived from them. We present our approach by "scaling up" a WebML application under a stress scenario, proving that it provides a "framework" for overcoming scalability issues.

1 Introduction

Scalability is the ability of a system to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth [3]. Scalability problems in the applications derived using Model Driven Web Engineering (MDWE) tools may not appear as soon as they are deployed, but rather after they has been "living" in production for some time.

In the Web engineering research area, MDWE approaches [11] have become an attractive solution for building Web applications by raising the level of abstraction and simplifying Web application development. Regardless of the approach used, the main focus is always the same: to create a Web application that satisfies functional requirements. As a consequence, little attention has been put to non-functional requirements such as scalability issues as they have been considered technological-dependent aspects [12]. Additionally, the Web applications derived from MDWE approaches pose a rigid/static architecture that cannot be easily changed hindering scalability fixes. To make things worse, diagnosing and fixing these problems becomes cumbersome and impossible to be done in the models thus forcing teams to deal with the generated code.

To overcome these problems, a recent study [7] has captured the top 20 problems that Web applications face to achieve scalability and it also shows that *"unforeseen scalability issues during development, can easily appear in a production environment"*. This list includes obstacles that affect both code and model based development and includes aspects such as monitoring, logging and caching. Furthermore, though scalability is a desired aspect of a Web application, achieving it does not only require the application to run fast (within an acceptable threshold), but it also involves other technical aspects that need to be handled during the application lifecycle [7].

Identifying a scalability problem (e.g. the application runs out of DB connections) can be easy, however *diagnosing* to find the root cause of the issue and *fixing* it consumes many human resources and it has been probed to consume 50%-75% of the software development effort [2]. We argue that the current status of MDWE tools can not help on that aspect as they do not allow to decompose the model elements until their primitives and as a consequence forces teams to deal with the generated code, thus breaking the models' abstraction and, consequently, losing the productivity improvement claimed by MDWEs.

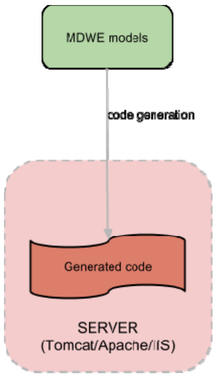


Fig. 1a. Traditional MDWE

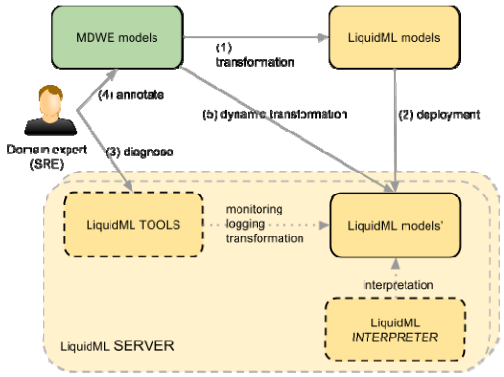


Fig. 1b. LiquidML approach

In this paper we present a model-based approach called *LiquidML* that complement MDWEs to help them diagnose and fix scalability issues in running applications. Instead of deriving the code from MDWE models (Fig 1a), we (Fig 1b):

1. Transform the MDWE models into LiquidML models that can be interpreted by our interpreter.
2. Deploy those models into a LiquidML Server (final platform for running them).
3. Provide tools for domain experts such as Site Reliability Engineers (SREs) to diagnose scalability problems directly at the model level after their identification.
4. Provide scalability annotations with architectural changes so the next derivation contemplates them to produce a more scalable version of the application once the problem has been diagnosed and as development continues in the higher-level MDWE models,
5. Enable to dynamically apply a model transformation to fix the problem at runtime directly in the LiquidML servers if we want to avoid a deployment (doing steps 1 and 2 again).

The paper is structured as follows: in Section 2 we present related work and in Section 3 we present the details of our approach. In Section 4, we show how the approach is implemented and in Section 5 we present the conclusions and further work.

2 Related Work

Developing and maintaining Web applications require not only an initial construction process but also a continuous monitoring/fixing cycles that must interleave with new requirements implementation. Scalability is generally an implicit desired feature by product owners but it is the least aspect that is paid attention on during development.

In this matter, the authors in [12] clearly states "*Many MDWE approaches have been created in the last 20 years with special focus in modeling the functional aspects of Web applications. Non-functional (e.g., scalability) properties of Web applications have traditionally been a minor concern in the Web engineering community and have been seen as technology or system-related issues*". Additionally, the author presents a theoretical proposal to deal with non-functional requirements during a model based development; however scalability is not treated as a specific concern and no implementation is presented.

Not restricting to Web Engineering, the aspects of dealing with scalability in Model-Driven Engineering (MDE) are the central topic of a recent study [8]. However, the term scalability in this area is treated not as the scalability of the derived application but to actually scale the tools and models to be able to handle relatively big applications. In this aspect, being able to scale the development of model based applications involves handling huge graph representing the models which compromise the performance of the transformations applied to obtain the applications.

In [5] the authors present an approach for transforming models into code while the application is running. To accomplish such task, the authors provide a runtime model with an API to push the code changes to the running environment. Though the approach makes sense from a conceptual point of view, many technological issues such as concurrency and well known issues in the derived language (e.g. memory issues in a Java program such as hot code replacement) are not taken into account. As a consequence, we argue that the approach, in its current state, lacks practical application.

Finally, holding models at runtime to perform runtime changes has been presented in [1]. However, the approach focuses in the representation of the actual requirements while the application is running rather than on the live models. The approach was initially applied for autonomous systems where domain experts are not able to access the system easily and thus has to run autonomously. The applications built with models @ runtime are from a different domain and seems to have less sophisticated business requirements than a Web application; as a consequence those automatic changes can be applied. In the Web engineering area, we think that those changes should be applied by SREs in a semi automatic way.

3 LiquidML

In this section we present our approach for fixing scalability issues at runtime. Though our approach can be applied to any MDWE method, we will illustrate it using WebML [4], as it is the most mature MDWE regarding to tool support.

3.1 LiquidML in a Nutshell

A LiquidML model is a composition of so-called *Flows*. A *Flow* describes a sequence of steps that need to happen within a Web request (called a *Message* in our approach).

A Message has a payload (body) and a list of properties while each step is visually identified by an icon and constitutes an *Element* of the flow. Communication between Elements happens by means of message interchanges. The way messages are moved from one Element to another one is defined by the Connections between them.

We have categorized the Elements using the categories found in [6]:

- Message source: Listens for incoming requests and generates messages from them. A concrete example is an HTTP listener that listens to incoming requests from users and creates Messages from HTTP requests.
- Processor: Processes a message by changing its payload and properties. Processors can vary from custom user logic to DB access.
- Router: Moves the message between processors depending on its type and conditions. For instance, a *ChoiceRouter* routes the message to a specific Element of its list, based on a Boolean condition.

To summarize the LiquidML concepts, a metamodel is presented in Fig. 2.

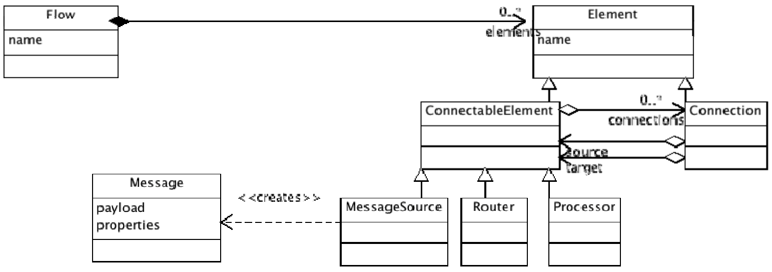


Fig. 2. LiquidML metamodel

3.2 Transforming MDWE Models to LiquidML Models

Flow models (main model of the our LiquidML models) could be manually created, however a more common scenario would be to start from higher level models (like WebML navigation models) and then transform those into our flow models to benefit from LiquidML. To provide an example of this transformation we show in Fig 3 the product details page of a WebML model. The product details page shows the basic product information and a rank that qualifies the product.

The actual transformation from the WebML model to LiquidML works iterating through the elements of the model and "unfolding" some elements that are implicit in the WebML model but that have an explicit representation in our lower level of abstraction (so that they can be configured, if needed, for scalability purposes). Fig. 4 shows the result of the transformation for the WebML example.

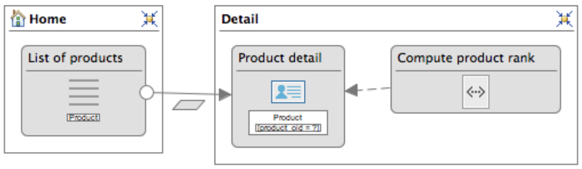


Fig. 3. Fragment of WebRatio model for our E-Commerce application

The Element with no incoming arrow represents the Message source listener that will receive incoming requests (implicit in WebML). The Element connected to the Message source named “Route path” is a ChoiceRouter that behaves like a choice/switch statement and it will route the message to the “Get info” processor if the request comes to a URL starting with “/product/*”. This element is represented in the Page units of WebML (“Detail” unit in Fig. 3) by a property that identifies its URL. The transformation also tries to derive highly concurrent models and thus the “Get info” is another router that gets information in parallel from multiple sources. It obtains the product info from the DB: (“Get product info”) and it triggers the computation of the product’s rank, which involves two database (DB) queries (“Get user reputation” and “Get product reviews”) and a Processor that computes the rank from this information (“Compute product rank”). The “Get product info” element is obtained from the “product detail unit” while the other elements are derived from the Groovy source code of “Compute product rank”. Finally, the information gets composed (“Compose data”) and used for rendering a Web page in the “Render template” processor. These elements are hidden in the WebML model as WebRatio generates an example UI; we also make these elements explicit.

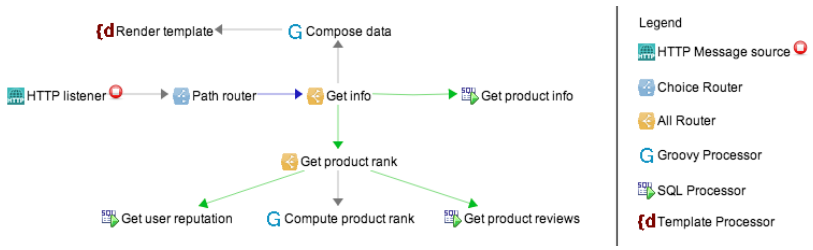


Fig. 4. Product details flow

A Flow model can be executed by means of the LiquidML interpreter. In the following subsection, we provide an overview of how our model interpretation works and a brief description of the deployment process.

3.3 Model Interpretation and Deployment

As aforementioned, flows define the behavioral part of the Web application. On the contrary to all MDWE approaches, we decided to interpret rather than to derive the code of a Web application. Strong cons and pros of both approaches can be found in [9] and in many informal discussions[1,2,3]; however, we do not expect to find a definite answer to this matter but rather present the advantages we found for Web scaling in our model based approach. It is true that at a first sight, code-generation seems to be

[1] http://www.theenterprisearchitect.eu/archive/2010/06/28/model-driven-development-code-generation-or-model-interpretation

[2] http://modeling-languages.com/executable-models-vs-code-generation-vs-model-interpretation-2/

[3] http://blog.webratio.com/?p=368

the right option when aiming at scalable applications but scalability involves more than an application running fast.

As our behavioral models (Flows) are rather simple, the interpreter algorithm is quite simple too. We present a simplified version of the algorithm using a Java-based pseudocode in the next lines: the interpreter works when messages are received (event based style [6]) on Message sources (e.g. an HTTP message source) (line 1). It finds the next element (currentElement) that will handle the message (line 2 and 5) and evaluates it using the message content (line 4). An evaluation returns a Message instance that could be the same as the previous one or a new one depending on the Element intent (data transformation, routing, etc.) and it is passed to the next Element until we run out of Elements (line 3).

```
1. OnMessageReceived(MessageSource msgSource, Message message): {
2.  Element currentElement = interpreter.getNextInChain(msgSource);
3.  while (currentElement != null) {
4.     interpreter.evaluate(currentElement, message);
5.     currentElement = interpreter.getNextInChain(currentElement);
6.  }
7. }
```

Interpretation happens while engineers are building the application and when the application is run in every other deployment environment (QA, Staging, Production). Once the models satisfy the requirements, the deployment process to a specific environment occurs. The deployment is an automatic process where a copy of the models is moved to the servers where they can start receiving messages. As aforementioned, unforeseen problems may appear in a production environment; thus in the following subsection, we present two tools to help diagnosing problems while our models are running.

3.4 Diagnosing Production Problems

The LiquidML model is what the engineering team tweaks and what is being run in production by means of the LiquidML interpreter. Such interpreter resides inside the LiquidML server, which is our execution platform that provides the ability of tweaking the models at runtime to improve the overall scalability of the application.

In our approach, each Element inside a Flow can suffer of production problems and as a consequence it may need to be adjusted. In these cases our approach allows us to monitor specific Elements but only when the engineer requests so. Monitoring adds a small performance footprint on each request that must be handled with caution. To diagnose problems we provide two well-known features such as profiling and logging. However, instead of generating low-level artifacts such as Java code, we work at a high level allowing to reference elements and messages.

As shown in [7], logging is a practical tool for this purpose as we can log information about the incoming messages. In our approach, we also allow condition logging (log information if it only satisfies some conditions). This is extremely important as it helps to reduce the performance degradation [10] of the elements when thousands of messages are processed per second and because it filters messages that we know are not causing the problem. Considering the Product detail example, we can dynamically add a logger to the “Path router” Element that prints the requested resources (the incoming URL) by simply writing the following expression:

```
‘Incoming traffic: ’ + message.properties[‘actions.http.requestURI’]
```

The second feature we provide is Element profiling, meaning that we allow the temporary measuring of the performance of an Element. It helps with the identification of elements that are consuming more time to complete in the Flow and orients modelers in providing quick performance solutions through the introduction of model optimizations. We can attach multiple profilers to our elements (Fig 5) and see how the average response time changes as traffic comes in. Using both logging and profiling we can better diagnose the problem and check if a runtime fix is possible or not; we will discuss this topic in the following subsections.

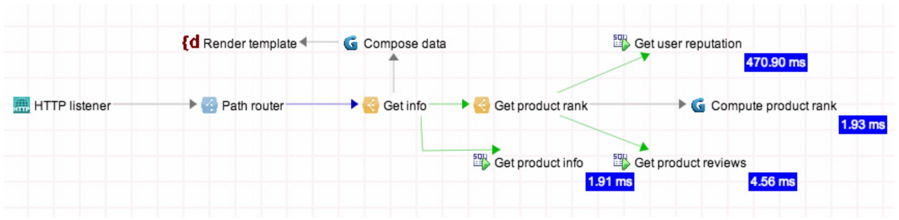


Fig. 5. Profiling a flow

3.5 Annotation of MDWE Models

Once the problem has been identified and diagnosed, the development team has to fix it. Our approach allows annotating the MDWE model with scalability annotations to improve the scalability properties of the applications derived from the MDWE tool. Fig. 5 shows that the computation of the rank is taking a large percent of the response time and as a consequence some optimization needs to be introduced. The development team proposes to cache the product rank that gets computed by tagging the navigation line in the WebML model of Fig. 3 with a cache strategy of the computed value. In Fig. 6 we show the tagged WebML model (the name ALC stands for “Async Lazy Compute” as shown in the following section).

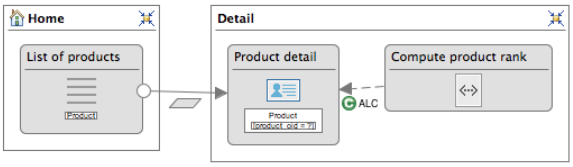


Fig. 6. Tagged WebML model

The tag applied to the WebML model generates a transformation that can be applied to the generated LiquidML model. In the following section, we present the result of applying such transformation.

3.6 Runtime Model Transformations

As mentioned in the previous subsection, some problems can be quickly fixed in our approach and do not require a build-deploy process. Those fixes tend to help satisfying non-functional requirements and thus functional requirements should not be altered. In this matter, we have created a small, non-exhaustive catalog of solutions that can be automatically applied to flows and it is based on the concepts found in [6]. Due to space constraints, we will not present the catalog but just explain the transformation applied in the example of Fig. 6.

The caching (Async Lazy Compute) is used when an Element is running slow, as a consequence we cache the results produced by the Element for a long period of time. We store when was the last time that the element was run and the result of such execution and upon incoming requests we update the cached results asynchronously. Applying this transformation to the connection between “Get info” and “Get product rank” will cache all the results coming from the subgraph starting in “Get product rank”. The transformation needs some inputs to be applied which can be automatically obtained if it is specified in the WebML model:

- A key expression: an expression used for the getting and putting the results in the cache. In our case we will use message. Properties ['actions.http.productId'].
- A value expression: what we want to cache. In our case just the payload of the message: message.payload.

The application of the transformation modifies the Flow model dynamically and leaves it in the state shown in Fig 7. We have highlighted the elements and connections added by the transformation, as the remaining of the diagram is the same as shown in Fig. 4. The first element added by the transformation “Get cached value” gets the cached value for this request using the product id. In the “Is cached?” router if it is not cached we continue to the “Chain” element. Here, we compute the values as before and then we store the value in the cache (“Put cached value”) and save the last update computation (“Put last update”). Finally, the “Async cache” gets asynchronously evaluated when the value has been cached before and (NOW() – lastComputedTime()) > configurable threshold. So, we recomputed the value in an asynchronous way but only when it is requested and it is old enough. The user may see an old value but after the new value has been computed it will appear for the following user.

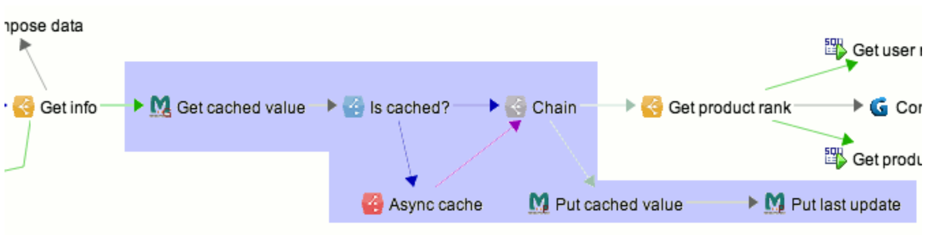


Fig. 7. Result of applying “Caching (async lazy compute)”

4 Implementation

We have developed some tools to support the construction of models using a Web based application that allows modeling each of the concepts we have mentioned. All the figures presented in this paper have been obtained from the Web tool we have built. Such tool allows handling multiple teams and each team allows the creation of multiple applications. Each application consists of multiple flows, properties and resources and it can be snapshot and deployed.

Once the application is deployed, we can move to a deployment view of the flows where we can modify the application at runtime to diagnose and fix any performance problem that may arise. In the deployment view, we can select an element and add a profiler. After some messages are received, we can start seeing the number of milliseconds it takes for a message to be processed. If necessary we can add some logging information that can help us diagnose the problem. The logging information is generated from a dynamic expression on the message and each recorded entry can be seen in the tool. Finally, if the problem can be fixed by simply applying a transformation, we can do so by selecting the Element involved, configuring the transformation (if necessary) and clicking in the apply button. The task is straightforward because the user selects a Flow Element used as a reference for the transformation and then chooses a valid transformation.

We have decided not to use the well-known Eclipse Modeling Framework (EMF)[4] for defining our language and its visual representation since, in our humble opinion, it provides a rigid structure that we want to avoid. As a consequence, our tool support required a bit of extra effort to be developed. To implement it we have used a standard Java architecture composed by Spring Framework[5] and Hibernate[6]. In addition, Cappuccino[7] was used to build the editor. On the server side, we implemented the REST API services with Jersey[8] so that the models can be deployed and share most of the Java code. We have also used Spring and Hibernate in the Server side too. We encourage the reader to visit our Web site at http://www.liquidml.com for demonstration videos.

5 Conclusions and Future Work

In this paper we have presented LiquidML, a model-based approach that complements MDWEs methodologies to help them improving the scalability of the applications they derive. To the best of our knowledge, this is the first work to propose and implement a solution to deal with this topic. Once a LiquidML model is obtained, we can monitor the application to help with the identification of production problems that can not be reproduced in any other environment. If the problem can be fixed at

[4] Eclipse modeling framework. `http://www.eclipse.org/modeling/emf/`
[5] Spring, `http://projects.spring.io/spring-framework/`
[6] Hibernate, `http://hibernate.org/`
[7] Cappuccino, `http://www.cappuccino-project.org/`
[8] Jersey, `https://jersey.java.net/`

runtime, engineers can apply a well-known solution safely and automatically, avoiding the high cost of redeploying the application.

Interpreting models poses multiple challenges and many advantages that we want to take into account as future work. From analyzing multiple empirical experiences, it is easy to abstract solutions to common patterns that appear in the application. Thus, we plan to provide these patterns that fix production problems and the rules that help their identification as first class entities. We predict that such approach cannot be fully automated as it requires knowledge about the running application (e.g. where the information is stored in the message and what are the things we want to cache) and as a consequence a semi automatic process using a rule engine seems to be a viable solution. Finally, we plan to expand the catalog of patterns that can be applied to flows, which will help SREs fixing more problems at runtime.

References

1. Blair, G., Bencomo, N., France, R.B.: Models@ run.time. Computer 42(10), 22 (2009)
2. Boehm, B.W.: Software engineering economics. Prentice-Hall, Englewood Cliffs (1981)
3. Bondi, A.: Characteristics of scalability and their impact on performance. In: Proceedings of the 2nd International Workshop on Software and Performance (WOSP 2000), pp. 195–203. ACM, New York (2000)
4. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): A Modeling Language for Designing Web Sites. Computer Networks and ISDN Systems 33(1-6), 137–157 (2000)
5. Sánchez Cuadrado, J., Guerra, E., de Lara, J.: *The Program Is the Model*: Enabling Transformations@run.time. In: Czarnecki, K., Hedin, G. (eds.) SLE 2012. LNCS, vol. 7745, pp. 104–123. Springer, Heidelberg (2013)
6. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, p. 735. Addison-Wesley (2012)
7. Hull, S.: 20 Obstacles to Scalability. ACM Queue 11(7), 20 (2013)
8. Kolovos, D., Rose, L., Matragkas, N., Paige, R., Guerra, E., Cuadrado, J.S., Lara, J., Ráth, I., Varró, D., Tisi, M., Cabot, J.: A research roadmap towards achieving scalability in model driven engineering. In: Proceedings of the Workshop on Scalability in Model Driven Engineering (BigMDE 2013). ACM, New York (2013)
9. Mellor, S.J., Balcer, M.: Executable UML: A Foundation for Model-Driven Architectures. Addison-Wesley Longman Publishing Co., Inc., Boston (2002)
10. Molyneaux, I.: The Art of Application Performance Testing: Help for Programmers and Quality Assurance, 1st edn. O'Reilly Media, Inc. (2009)
11. Rossi, G., Pastor, O., Schwabe, D., Olsina, L.: Web Engineering: Modelling and Implementing Web Applications. Springer (2007)
12. Toffetti, G.: Web engineering for cloud computing. In: Grossniklaus, M., Wimmer, M. (eds.) ICWE Workshops 2012. LNCS, vol. 7703, pp. 5–19. Springer, Heidelberg (2012)

Multi Matchmaking Approach for Semantic Web Services Selection Based on Fuzzy Inference

Zahira Chouiref[1,2], Karim Benouaret[3], Allel Hadjali[2], and Abdelkader Belkhir[4]

[1] Université de Bouira, 10000 Bouira, Algeria
zahira.chouiref@univ-bouira.dz
[2] LIAS, ENSMA, 86360 Chasseneuil-du-Poitou, France
{zahira.chouiref,allel.hadjali}@ensma.fr
[3] LT2C, Université Jean Monnet, 42000 Saint-Etienne, France
karim.benouaret@univ-st-etienne.fr
[4] USTHB, 16000 Algiers, Algeria
kaderbelkhir@hotmail.com

Abstract. Selecting services from those available according to user preferences plays an important role due to the exploding number of services. Current solutions for services selection focus on selecting services based either only on non functional features, on context preferences or profile preferences. This paper discusses an improvement of existing services selection approaches by considering both user context and profile. The ultimate aim is to derive maximum profit from available profile and context information of the user by inferring the most relevant preferences w.r.t his/her contextual profile. Linguistic/fuzzy preference modeling and fuzzy inference based approach are used to achieve efficiently a selection process. Some experiments are conducted to validate our approach.

Keywords: Web Services Selection, Profile, Context, Preferences, Fuzzy Logic Theory, Fuzzy Inference Rules, Contextual Profile Matching.

1 Introduction

Semantic Web services; *SWS* field plays an increasingly important role in enhancing the user interaction in the Web and enterprise search, as well as in providing a flexible solution to the problem of application integration. With the rapid worldwide deployment of offered services on Internet, SWS selection has been an active and fast growing research area. The services selection is a technique which uses functional features *(Input, Output, Precondition, Effect; IOPE)* and non functional features *(Quality of Service; QoS, etc.)* [12], to search Web services from large scale service repositories that fit best the user requirements. Development of methods which would increase research accuracy and reduce research time is one of the main challenges in SWS selection. Most of these approaches focus on satisfying the functional requirements. A service consumer copes with a difficult situation in having to make a choice from a mass of already discovered services satisfying the functional requirements. To discriminate such discovered services, the focal point of current SWS selection is on the non-functional aspect of a service [12]. This can be done using *QoS parameters* [1], *context* [8], *preferences* [11], *profile* [5]. These approaches help to improve the service discovery, selection and composition and simplify the management process for non functional attributes of Web services. However, such approaches do not address the issue of: i) Taking into account all the information characterizing the service (offered and requested) often called contextual profile; *CP*, ii) the gradual nature of the parameters related to context/preferences in a human language, iii) deriving new relevant preferences on the basis of user *CP* information by means of fuzzy inference rules. The key concept of the approach is the user/service profile where fuzzy logic theory is used to describe information related to the profile in a faithfully way. The first objective of this paper is to propose a common profile model that can capture all information describing the user and the service. The second objective is to introduce linguistic terms to express preferences and fuzzy rules to model contextual preferences.

The remainder of this paper is structured as follows: Section 2 presents a brief background on fuzzy set theory and provides a survey on existing approaches of SWS selection. In Section 3, we set up a required and provided service model based on *CP*, then we provide an SWS selection framework based on our model. Section 4 describes the query processing in a real case study related to the field of restaurant business and the ranking mechanism as well. In Section 5, an experimental study is described to show the feasibility and effectiveness of our proposal. Finally, Section 6 concludes the paper.

2 Related Work

A key issue in service computing is selecting service providers with the best user desired quality. Recently, existing service selection approaches reviewed below are distinguished by the fact that they rely on QoS parameters [1], context information [8], user preferences [2][7][9][11], etc.

A matchmaking algorithm proposed by Adnan et al. [1] is based on tying QoS metrics of Web service with fuzzy words that are used in users request. The aim of the paper is to satisfy user's requirements and preferences regarding only QoS and not all preferences related to non-functional service parameters. In [8], authors proposed the non-functional properties that are related to local constraints which reflect the user preferences and context of the demanded service. However, in real-life systems, context information is naturally dynamic, uncertain, and incomplete, which represents an important issue when comparing the service description with user requirements. This approach, however, is based on context rather than the data of services and could not handle both exact and fuzzy requirements. They do not allow reasoning on context information to determine criteria weights automatically, also the model proposed by the authors does not consider the implicit user preferences. Web services selection based on preferences mainly consider single user's preferences. Benouaret et al [2] introduce a novel concept called collective skyline to deal with the problem of multiple users preferences to select skyline Web services. Chao et al. [4] proposed a framework, which leverages fuzzy logic to abstract and classify the underlying data of Web services as fuzzy terms and rules. The aim is to increase the efficiency of Web services discovery and allow the use of imprecise or vague terms at the level of the search query. Steffen et al [7] model Web service configurations and associated prices and preferences more compactly using utility function policies, and propose flexible and extensible framework for optimal service selection that combines declarative logic-based matching rules with optimization methods, such as linear programming. In [10], a framework of SWS discovery based on fuzzy logic and multi-phase matching is proposed in this work. The first level matchmaking is executed with service capability against the second level matching is executed with service fuzzy information. The authors do not take into account neither the vague information of the user profile (personal information, etc.) nor the vague information of the context.

The presented work links user profile, user context and user preferences and provides suitable selection method that uses inferred preferences in order to have powerful, yet scalable ranking process. These preferences are inferred from the user's contextual profile, this is done by directly applying an efficient inference method based on *an extended modus ponens.*

3 Our Model

3.1 Service Description Model

We describe a SWS (provided / required) by the following model, which not only supports service capability information (IO), but also supports *service profile*, *service context* and presents *service vague information* associated to *preferences.* For the sake of illustration, the following reference example is used.

Reference Example: Let U be a user wants to book a hotel in Australia. He sets his preferences and submits the following query Q: "return the hotels in Australia preferably {***near*** **to his position**} with {***affordable*** **price**} and {***at least three*** **stars**} and having a **restaurant** with an {***asiatic*** **cuisine**}, knowing that he has a **car** and he is accompanied by his **wife** and his **child**".

Definition 1 (Advertised Service Description Model/Required Service Description Model). An advertised Web service (A required Web service) is described by the following model respectively: $SA = \{CA,\ CPA,\ FZA\}/$ $SR = \{CR, CPR, FZR, \theta\}$, where:

– CA/CR is the advertised/required service capability information description, which contains *(NA, DA, FPA)/(NR, DR, FPR, θ)* where NA/NR is the name of the advertised/required service, DA/DR is the functional description of the advertised/required service and FPA/FPR is all functional parameters of the advertised/required service *(IOPE).*

- *CPA* = $(CPA_1, \ldots CPA_n)$, *CPR* = $(CPR_1, \ldots CPR_n)$, is a set of non-functional parameters that make up the *CP* of *SA/SR*. The detail context can be explored in section 3.2.
- *FZA/FZR* is a set of linguistic terms which used to describe the vagueness that pervades information containing in *CA/CR* and *CPA/CPR*.
- $\theta(0<\theta<1)$ is a threshold such that if the satisfaction degree of a service w.r.t the query at hand is below this threshold, it is not retrieved.

Each I/O attribute of request a_i is characterized by a set of preferences values p_i, i.e. (a_i, p_i). For each SA attribute, we can assign crisp constraints (e.g. MinStars) or fuzzy constraints (FZA) (e.g. CheaperPrice). The attributes are self explanatory, which indicates the pereference values which help the user to choose the service that suits its preferences. For each SR attribute, we can assign crisp preferences, or fuzzy preferences (FZR) (e.g. AffordablePrice).

3.2 Fuzzy Model to Contextual Profile

Fuzzy Contextual Profile Modeling. The information of *CP* can be: static such as *Profile* {*Personal data, etc*}, evolutionary such as *Preferences* {*Colour, Language, etc*} and temporary such as *Context* {*Devices, Localization, etc*}. These pieces of information must be captured to match demands to offers of services, in order to improve the relevance of answers during a selection process. For a given query *X*, we define its *CP* environment CPE_x as a finite set $\{(P_1, P_2, \ldots P_n)\}$ of multidimensional parameters, for instance, {*personal informations, context, preferences, etc*}. Each parameter *Pi* is modeled as a finite set $\{(C_1, \ldots C_m)\}$ of concepts, for instance {*demographic information, spatial context, display mode, etc*}. Each concept *Ci* is modeled as a finite set $\{(C'_1, \ldots C'_t)\}$ of sub concepts for instance {*gender value, family situation, country name , etc*}, and/or a finite set $\{(v_1, \ldots v_t)\}$ of attributes value, for instance {*full screen, postscript format, etc*}. Each concept (sub-concept) is characterized by a set of *preference values.* The attribute value domain $dom(C_i)$ can be expressed by means of: numerical assessments, logical assessments and fuzzy linguistic assessments.

An instantiation of the *CP*, called *CP state*, writes:
$w = (C_1 \text{ is } v_1 \wedge \ldots \wedge C_k \text{ is } v_k),\ k \preceq m$, where each $C_i \in CPE_x$, $1 \preceq i \preceq k$ and $v_i \subseteq dom\ (C_i)$ (the symbol $\wedge$ denotes a conjunction).

Example 3. For instance, *w* may be (family situation is *married*, means of transport is *car*, accompanying people is *wife* and *child*) for the example above.

Definition 2 (Contextual Profile Preferences). *A contextual profile preference CPP is a fuzzy rule of the form:* ***if*** C_1 ***is*** $v_1 \wedge \ldots \wedge C_m$ ***is*** v_m ***then*** A_1 ***is*** $F_1 \wedge \ldots \wedge A_l$ ***is*** F_l*, where vi, $1 \leq i \leq m \leq n$, stands for a crisp or fuzzy value of the context or the profile parameter CP_i and F_j, $1 \leq j \leq l$ represents a fuzzy preference related to attribute A_j.*

The meaning of *CPP* is that in the *CP* state specified by the left part of the rule, the preference A_j *is* F_j is inferred. From the user profile, one can deduce the following preferences on the searched hotels:

Example 4. A user who has a *car*, generally prefers hotels with *parking*. This may be expressed as (CPP_1): **if** *means of transport is car* **then** *PreferenceParking is yes.*

User Preferences Modeling. Let us now discuss the notion of fuzzy preferences: for instance, *"affordable price"* and *"nearest city"* are primitive terms. A primitive term can be described thanks to fuzzy sets, allowing to obtain for a price and a given distance, the satisfaction levels defined on the interval [0, 1]. As for categorial attributes, the membership functions are modeled as follows: The membership function of *"CuisineStyle"* is modeled by: $\mu_{C\_cuisine}$={1/chinese, 0.9/japanese, 0.8/thaiwanese, 0.7/sushi, 0.7/indonesian, 0.5/vietnamese, 0.3/indian, 0.2/pakistani, 0.1/americain} for chinese cuisine and $\mu_{F\_cuisine}$={1/french, 0.9/belgian, 0.8/mediterranean, 0.7/italian, 0.7/dutch, 0.6/latin, 0.5/german, 0.4/british, 0.2/american} for french cuisine. The membership function of parking is $\mu_{parking} = \{1/yes, 0/no\}$.

4 Query Processing

4.1 Semantic Web Service Selection Framework

Let H_D be a Hotels database. The desired services should accept {Address} as inputs and return {HotelName, StarsNumber, Price, CuisineStyle} as output for the case of our *reference example*. This query Q is written as follows:
SELECT *name-Hotel* **FROM** H_D **WHERE** (Hotel.Price is *affordable* AND Hotel.Stars is *at least 3* AND Hotel.Dist is *near of city* AND Hotel.CuisineStyle is *asiatic*).

Fig. 1 gives an overview of the selection framework. The matching engine must match the list of services with the input, output specified by the user. The main steps of the framework are:

First Search Filter. The result returned by *step 1* and *2* of Fig. 1 is $\sum_Q$, the list of services that correspond to the desired *number of stars*, *price*, *asiatic cuisine* and *distance to user*. Note that the preferences expressed on these attributes are mandatory. The system will computes the distance or similarity between all the concepts vector of the query and those of the Web services by means of suitable measures of similarity. In presence of different types of attributes, for each attribute, an adequate similarity measure is used as the following functions:
S_{p1}, S_{p2}, S_{p3} and S_{p4} represents respectively functions that compute the degrees of satisfaction of the *distance*, *Price*, *Number of Stars* and *Cuisine Style* of the SWS at hand w.r.t the fuzzy set modeling the user preference on attribute *distance*, *Price*, *Stars* and *Cuisine*. For instance, $\mu_{near}(8) = 0.4$, $\mu_{affordable}(45) = 0.5$, $\mu_{stars}(4) = 1$ and $\mu_{F\_Cuisine}(italian) = 0.7$.

It is worth noticing the all the functions $(S_{p1}, S_{p2}, S_{p3}, S_{p4})$ *provide degrees that belong to the same scale [0,1]. This property of commensurability allow aggregating them, in a convenient way, to obtain an overall score of an SWS.*

Overall Matching $Score_1$**.** After calculating the different individual matching values of each attributes of a service, one way to obtain an overall matching score is to aggregate these individual matching values using a *T-norm* operator (such the *min operator*) as follows:
$Score_1 = \top(S_{p1}, S_{p2}, S_{p3}, S_{p4}) = \min(S_{p1}, S_{p2}, S_{p3}, S_{p4})$.
Now, if the $Score_1$ of a service is higher than θ, then this service is added to $\sum_Q$ list of services that satisfying request's attributes.

Second Search Filter. In this phase, not only *I/O* parameters has to be considered but also contextual profile information such as accompanying people, age of child, etc. This means that service contextual profile should contribute to the development of an advanced search strategy. The steps (3, 3', 4, 5, 5', 6) of the Fig. 1 illustrate this strategy and are summarized in the following: (i) Infering a set of relevent preferences and their semantics from the fuzzy rules base B^{CPP1}, regarding the user *CP* state *w*, then augment the query by the inferred preferences. To achieve this, we make use of a *knowledge based model* described bellow. (ii) Calculation of the satisfaction of the result provided by the first filter w.r.t. to the inferred preferences.

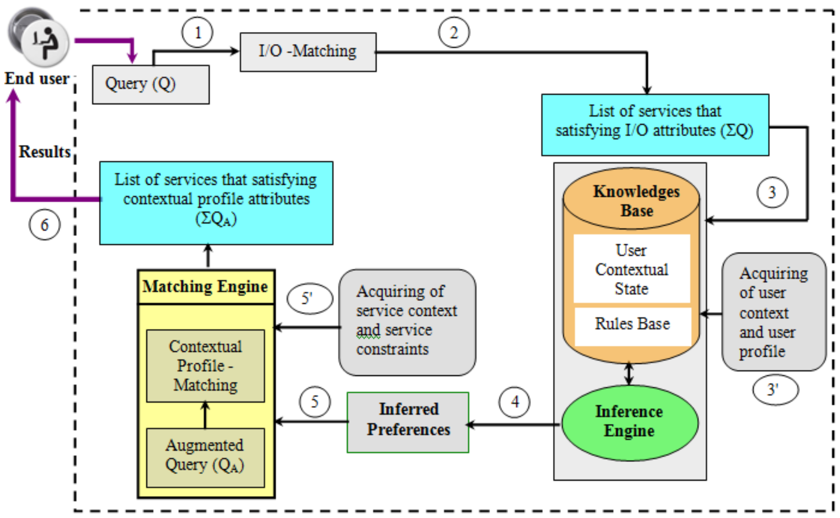

End user
Query (Q)
1
I/O -Matching
2
List of services that satisfying I/O attributes (ΣQ)
3
Knowledges Base
User Contextual State
Rules Base
Acquiring of user context and user profile
3'
Inference Engine
4
Inferred Preferences
5
Matching Engine
Contextual Profile - Matching
Augmented Query (QA)
5'
Acquiring of service context and service constraints
List of services that satisfying contextual profile attributes (ΣQA)
Results
6

Fig. 1. Semantic Web Services Selection Framework

[1] Is a set of contextual profile preferences that can be built from users' experiences in the domain considered.

Knowledge Based Model. We rather propose here to specify contextual preferences by means of *fuzzy gradual rules*. The use of gradual rules for deriving preferences with respect to the user *CP* leads to refining the search results and returns the best ones to the user.
Inference Process: We assume available $B^{CPP} = \{CPP_1, ..., CPP_m\}$ a fuzzy rules base modeling a set of contextual profile preferences. Each rule is of the form **if** C_1 **is** $v_1 \wedge ... \wedge C_m$ **is** v_m **then** A_1 **is** $F_1 \wedge ... \wedge A_l$ **is** F_l (see Definition 2.). Now, given a user's *CP* state, i.e., *CP*=(C_1 **is** $v_1 \wedge ... \wedge C_k$ **is** v_k), one can derive relevant preferences to the user by using a fuzzy inference schema, called the *Generalized Modus Ponens; GMP* [6]. In a simple case, from the rule: **if** *C* is *V* **then** *A* is *F* and the fact: *C* is *V'*, where *V*, *F* and *V'* are gradual predicates modeled thanks to fuzzy sets, the *GMP* allows inferring the preference *A is F'* and the fuzzy semantics of *F'*. See [6] for more details.

Example 6. The preference *Parking is Yes* is inferred by applying the rule (**If** user has a *car* **then** preferenceParking is *Yes*) and the fact (user has a *car*). Note that *car* and *yes* are fuzzy sets represented by (car/1)(yes/1) respectively.

Example 7. In our booking hotels example, assume available the rules base B^{CPP} of Table 1. The preference *Hotel is Animated'* is inferred by applying the rule (**If** age is *young* **then** hotel is *animated*) and the fact (age is *about_26*) where *young* and *about_26* are fuzzy sets represented by (0,0,25,27) and (24, 26, 26, 28) and *animated* is expressed using a qualitative rating such as: *low*, *medium*, *high* and *very high* whose semantics are given by triangular membership function. The semantics of *Animated'* is calculated from the semantics of *Animated, young* and *about_26*, see [3].

Our model entirely leverages the knowledges base presented in Table 1 to derive new relevant preferences. A rule-based representation which includes different possible preferences for our reference example is proposed.

Augmented Query Process: Once the user preferences are inferred, we offer the system the possibility of augmenting the query in order to refine the selection process. Then, we have a final user query Q^A, the augmented query of Q writes: $Q^A = \{C_1 \wedge C_2 ... \wedge C_n \wedge P_1 \wedge P_2 \wedge ... P_m\}$.

Example. Q^A = {*Price* ∧ *Stars* ∧ *Distance* ∧ *CuisineStyle* ∧ ***Parking*** ∧ ***Hair and Beauty Service*** ∧ ***Sauna*** ∧ ***Games and Activities*** ∧ ***Kid Friendly Menu*** ∧ ***Childrens Highchairs*** ∧ ***Family*** ∧ ***Animated'***}, where the infered preferences appear in bold.

Similarity Computing: in this phase, the system computes the similarity between inferred user preferences and services' constraints by means of fuzzy semantics associated w.r.t these preferences. To accomplish this phase, the system will evaluate the similarity degree between all the inferred preferences from B^{CPP} and the services' constraints of $\sum_Q$ by using the following functions: Now,

Table 1. Knowledges Base

Knowledges Base		
Rule	**Rules Base**	**Facts Base**
R1	**If** user has car **then** preferenceParking is Yes	Car
R2	**If** accompanying people is wife **then** facilities is hair and beauty service	Married
R3	**If** accompanying people is wife **then** facilities is sauna	Wife
R4	**If** accompanying people is child **then** facilities is games and activities	Child
R5	**If** accompanying people is child **then** facilities is kid friendly menu	
R6	**If** accompanying people is child **then** facilities is childrens highchairs	
R7	**If** accompanying people is wife **then** theme is family	
R8	**If** age is young **then** hotel is animated	

assume that the initial query$Q$ is only augmented by the inferred preferences: *Parking* and *Animated'*. To compute the satisfaction of each hotel h $\in \sum_Q$ w.r.t such preferences, we use:
S_{p5}: for each h $\in \sum_Q$, we have $\mu_{parking}(h) = 1$ if *h.parking = yes*, 0 otherwise, and S_{p6}: the fuzzy semantics of the predicate *Animated'* is computed by means of the combination/projection principle [3] (see example 7). Then, for each h $\in \sum_Q$, the degree of satisfaction is $\mu_{animated'}(h)$.

Overall Matching $Score_2$.
$Score_2 = \top(Score_1, S_{p5}, S_{p6}) = min(Score_1, S_{p5}, S_{p6})$.

Overall Score S. We use an aggregation function of $Score_1$ and $Score_2$ to compute the overall score S. Now, to give priority to the initial preferences w.r.t. to inferred preferences (IP) we make use of the following formula (where $\alpha \in]0, 1]$ is the priority of IP). Then S is given by:
$S = min(Score_1, max(Score_2, 1 - min(Score_1, \alpha)))$
Finally, the user can select the top-k answers or the answers whose score S is greater than a given threshold.

5 Experimental Evaluation

The main purpose of this evaluation is to compare the effectiveness of our proposed selection framework (referred to as IP for inference process) with the traditional frameworks that do not use the inference process (referred to as TR for traditional). We perform a case study, due to the limited availability of public services. We created a set of 100 synthetic restaurant service descriptions, and we involved different users to conduct our experiments. However, due to lake of space we only report results regarding 4 users.

Fig. 2 shows the precision of IP and TR at various ranks for 4 different users. Observe that IP has consistently better precision than TR since IP includes into the ranking process inferred preferences that are interesting for users. See also that, for user$_1$ and user$_2$ IP has an almost perfect precision, while the precision of TR is mediocre. Moreover, for user$_3$ and user$_4$ IP and TR have similar precision at rank 15 and rank 20. The reason is that the increase of the rank may increase the probability that similar services belong to the top-k list of both approaches.

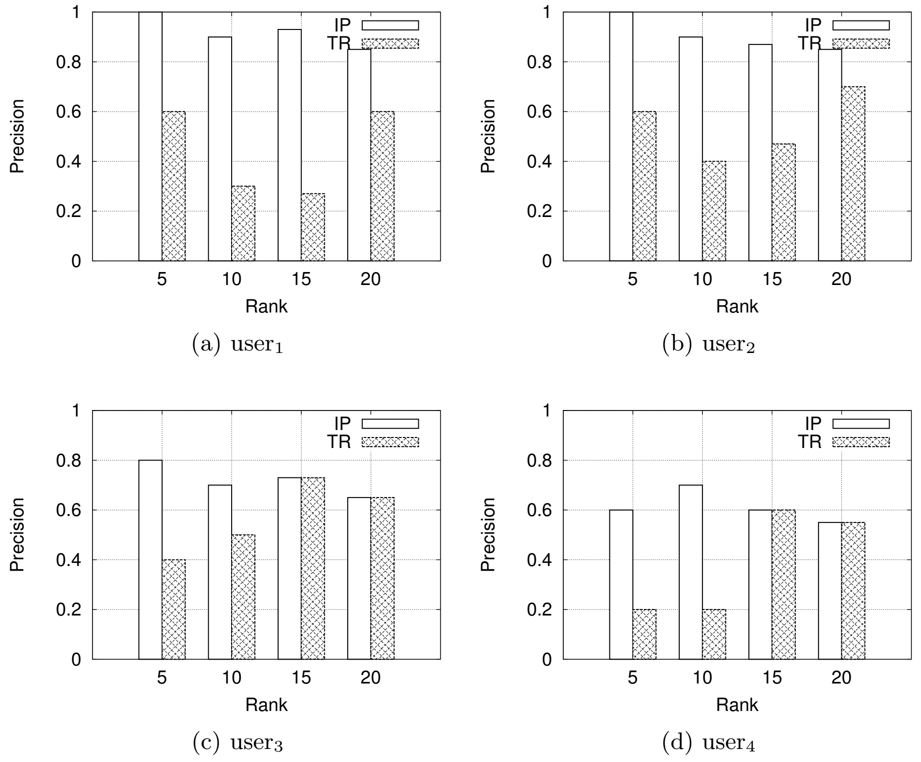


Fig. 2. Precision at Rank ($\theta = 0.5$, $\alpha = 0.5$)

6 Conclusion

This paper has proposed an SWS selection framework based on user (fuzzy) preferences. The goal of our work is to enhance accuracy of SWS search results by using multi matching level to compute similarity between advertised SWS and users request and taking into account the user preferences that may be inferred from users profile. We also showed the interest of using gradual rules for representing contextual preferences and deriving new preferences that are relevant to the user. The proposed framework makes the integration of user contextual profile and fuzzy inference rules techniques into the selection process. Some experiments are done to show the feasibility and the precision of our proposal.

References

1. Al Rabea, A.I., Al Fraihat, M.M.: A new matchmaking algorithm based on multi-level matching mechanism combined with fuzzy set. Journal of Software Engineering and Applications 5(3) (2012)
2. Benouaret, K., Benslimane, D., HadjAli, A.: Selecting skyline web services for multiple users preferences. In: ICWS, pp. 635–636 (2012)

3. Bouchon-Meunier, B., Dubois, D., Godo, L., Prade, H.: Fuzzy sets and possibility theory in approximate and plausible reasoning. In: Fuzzy Sets in Approximate Reasoning and Information Systems (1999)
4. Chao, K.M., Younas, M., Lo, C.C., Tan, T.H.: Fuzzy matchmaking for web services. In: 19th International Conference on Advanced Information Networking and Applications, AINA 2005, vol. 2, pp. 721–726. IEEE (2005)
5. Chouiref, Z., Belkhir, A., Hadjali, A.: Advanced profile similarity to enhance semantic web services matching. International Journal of Recent Contributions from Engineering, Science & IT (iJES) 1(1), 1–13 (2013)
6. Hadjali, A., Mokhtari, A., Pivert, O.: A fuzzy-rule-based approach to contextual preference queries. In: Computational Intelligence for Knowledge-Based Systems Design, pp. 532–541. Springer (2010)
7. Lamparter, S., Ankolekar, A., Studer, R., Grimm, S.: Preference-based selection of highly configurable web services. In: Proceedings of the 16th International Conference on World Wide Web, WWW 2007, pp. 1013–1022. ACM, New York (2007), http://doi.acm.org/10.1145/1242572.1242709
8. Reiff-Marganiec, S., Yu, H.Q.: An integrated approach for service selection using non-functional properties and composition context. In: Handbook of Research on Service-Oriented Systems and Non-Functional Properties: Future Directions, pp. 165–191 (2011)
9. Skoutas, D., Alrifai, M., Nejdl, W.: Re-ranking web service search results under diverse user preferences. In: VLDB, Workshop on Personalized Access, Profile Management, and Context Awareness in Databases, pp. 898–909 (2010)
10. Su, Z., Chen, H., Zhu, L., Zeng, Y.: Framework of semantic web service discovery based on fuzzy logic and multi-phase matching. Journal of Information and Computational Science 9, 203–214 (2012)
11. Wang, H., Shao, S., Zhou, X., Wan, C., Bouguettaya, A.: Web service selection with incomplete or inconsistent user preferences. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 83–98. Springer, Heidelberg (2009)
12. Yu, H.Q., Reiff-Marganiec, S.: Non-functional property based service selection: A survey and classification of approaches (2008)

Semantic Mediation Techniques for Composite Web Applications

Carsten Radeck, Gregor Blichmann, Oliver Mroß, and Klaus Meißner

Technische Universität Dresden, Germany
{carsten.radeck,gregor.blichmann,oliver.mross,
klaus.meissner}@tu-dresden.de

Abstract. The mashup paradigm allows end users to build custom web applications by combining data-exchanging components in order to fulfill specific needs. Since such building blocks typically originate from different third party vendors, compatibility issues at component interface level are inevitable. This decreases re-usability and requires skilled users or automatisms to provide the necessary mediation to solve such issues. However, current mashup proposals are very limited in this regard.

We present techniques for data mediation that leverage semantically annotated interface descriptions to overcome a high degree of interface mismatch. We equipped the EDYRA mashup platform for end user development with automatic support for these techniques to increase the re-usability of components and to foster the long tail of user needs. In order to show the practicability of our approach, we describe the platform implementation and present benchmark results.

Keywords: mashup, semantics, data mediation, end user development.

1 Introduction

Recently, universal composition approaches like CRUISe [1] allow for platform-independent modeling of mashups and a uniform description of components spanning all application layers. Since components are typically developed by different third party providers, combining component interfaces in a meaningful way is far from trivial. Data exchanged between components may differ in various aspects leading to incompatibilities: providers use different vocabularies, schemata, units or abstraction levels when designing interface signatures. This complicates end user development (EUD) further, and connecting components in ways not anticipated becomes a cumbersome task. Semantic technologies are a potent solution to provide **data mediation**, i. e., automatic resolving of heterogeneous data structures. Although proposals in the semantic web service (SWS) domain exist, most mashup platforms neglect data mediation so far.

Within the EDYRA project, we adhere to universal composition and strive for enabling domain experts without programming skills to build and reuse composite web application (CWA). We utilize semantic annotations to refer to ontology concepts of component interfaces. Based on this, semantic data mediation techniques are applied by our platform and hidden from end users.

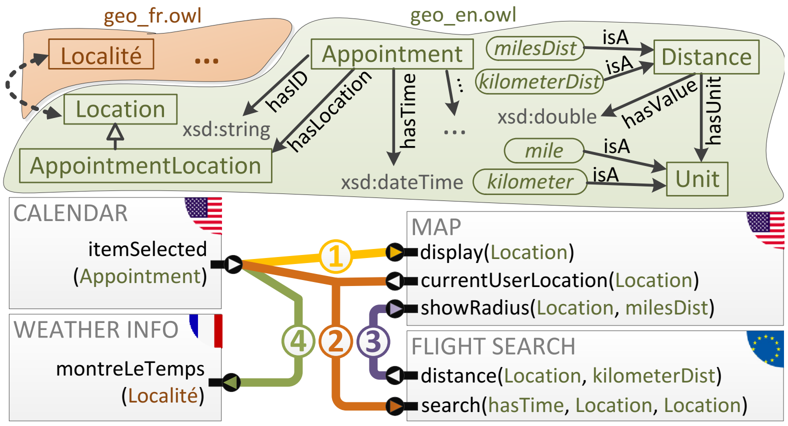


Fig. 1. Reference scenario: conference planning

To highlight arising challenges, let us consider the following use case which we use as a **reference scenario** throughout this paper.

Non-programmer Bob from the USA builds a CWA (Fig. 1) to organize a conference participation in Toulouse, France. He selects the conference, which he noted previously as an appointment, in the calendar. Then Bob wants to see the appointment location on a map. However, syntactically there is no possibility to combine the calendar and the map component's interface. While the calendar offers a data object of type `Appointment`, the map consumes a `Location`. Therefore, semantically it would be possible to take the appointment's `AppointmentLocation` and "cast" it to a general `Location` ①. Next, he wants to search for flights to Toulouse ②. In this case, he adds a flight search service and uses his current location as well as the appointment's location and time as search criteria. Besides the semantic problem of querying time and location of the appointment, it is necessary to put all three parameters together in compliance with the signature required by the flight service. Furthermore, Bob wants to visualize the distance between the airport and the conference location, which is calculated by the flight service ③. Because the flight service is located in Europe, the distance is provided in kilometers. To use the `showRadius` functionality of the map (from an American provider) this value has to be converted to miles. Finally, Bob wants to see the weather forecast for the appointment's time and location. He utilizes a French weather service which uses an ontology for annotating the interface that differs from the calendar's ④. But semantically the same concepts are described and the components can be coupled.

Current mashup proposals lack capabilities to implement this scenario. Thus, as our main contribution, we introduce data mediation techniques for CWAs and show their practicability within our EUD platform. These concepts help to combine components in more flexible ways than pure syntactic interfaces would allow, increasing re-usability and fostering the long tail of user needs.

The remaining paper is structured as follows. Sect. 2 presents mediation techniques for CWA. In Sect. 3, we describe our mashup platform with mediation support and show the practicability of our concepts. In Sect. 4, we discuss related work. Sect. 5 concludes the paper and outlines future work.

2 Semantic Data Mediation Techniques for CWA

Data mediation serves to resolve interface incompatibilities, of course within certain boundaries. In this section, incorporating results from the SWS domain like WSMO Mediators [6], we introduce a set of generic mediation techniques for CWA. We apply semantic data mediation and thereby leverage the domain knowledge defined in OWL-DL ontologies and annotated to component interfaces [1]. Essentially, annotations refer to classes, datatype and object properties or individuals in ontologies modeling the application/component domain. Since it is possible to model the same domain in various ways, we assume a certain modeling and annotation style.

In general, ontology classes can be annotated directly, e. g. `Location`, or via an OWL object property whose range the class is, if it is necessary to highlight a more specific meaning, e. g. `hasCenter`. Additionally, OWL datatype properties can be used, e. g. `hasLatitude`. However, there are circumstances where it is more appropriate to model individuals rather than subclasses. Units, currencies, and quantities, i. e., convertible concepts, may be mentioned as examples, or classes that refer to such convertibles on OWL property level (see `Distance` in Fig. 1). In this case, concrete individuals should be annotated, e. g. `milesDist`, a `Distance` individual where `hasUnit` points to `mile`.

In general, data transfer is realized through interface elements, which are *properties*, *operations* and *events* in our case. Interface elements can have one (e. g. properties) or more (e. g. operations and events) parameters, which have an identifier and a semantic type annotation. Channels combine one interface element of a source component SC with one of a target component TC. Therefore, an assignment $assi$ has to exist, that maps n parameters P_{out} of the SC bijectively to all n parameters P_{in} of the TC. A perfect match exists if $assi$ only includes mappings between parameters that are semantically identical (both refer to the identical concept). As an example, the mapping `(latitude, longitude)` $\rightarrow$ `(latitude, longitude)` is a perfect match.

Due to the usage of third-party components, a perfect match is unlikely. P_{out} and P_{in} can be *semantically compatible* if a **Semantic Connector** $SeCo$ can be defined, which is a set of channels and mediation techniques. It ensures that all parameters P_{in} of *one* interface element of a TC are connected and of the required semantic type. This may include that several channels from one or more SC can exist. In case of multiple inbound channels, the $SeCo$ takes care of an appropriate synchronization between them.

Upcast. As proposed earlier [1], the *upcast* mediation technique serves for solving different generalization levels of concepts annotated at parameters. In case of classes this means, that a more specific class is cast into a more generic one as long as they are in `subClassOf` relationship. Assume that a component outputs an `AppointmentLocation` but the target component requires a more generic `Location`. Then a upcast can be applied.

In principle, upcasts may additionally be used for OWL object properties if there is a `subPropertyOf` relation, by dealing with it as if the range would have been annotated. In case of datatype properties we presume that the underlying range stays the same, rendering upcasts simple. Upcasts are one way, i. e., only casts upwards the inheritance hierarchy are valid along the data flow.

Conversion. The *conversion* mediation technique has two main application areas. First, it resolves incompatibilities between two parameters annotated by convertible concepts, like *units, quantities* and *data types*. Please consider the example `kilometer` → `mile` from Fig. 1. The specific knowledge required can, e. g., be formalized in dedicated ontologies like the QUDT (`http://qudt.org`). In the latter, a base type is assigned to each unit for conversion purposes. For example, `meter` is the base type for `LengthUnit`, to which all other length units have a conversion factor to. Another use case are *scale adjustments* requiring more domain-specific transformations, e. g., mapping a five star rating to a ten point rating. Typically, these conversions require domain-specific knowledge and cannot be covered by generic algorithms or reasoners.

Second, conversion is used on class level in case of `equivalentClass` relationships, e. g. `Location` and `Localité` in Fig. 1 or `Location` and `Place`. For sake of simplicity, we pose a rather strict definition of equivalence: There have to exist `equivialentProperty` relations for all declared properties of those classes.

Semantic Split. A *semantic split* queries multiple OWL properties of an individual, which is represented as a parameter or property, and distributes them on one or more parameters of a target interface element. Fundamentally, only individuals can be "split" within the restrictions of their ontology class. This mediation technique is applicable if the following OWL constructs are annotated:

- Class: OWL data and object properties can be assigned to target parameters that reference a semantically compatible class, data or object property, e. g. connection ① in Fig. 1 (`Appointment.hasLocation → Location`).
- OWL object property: This case is handled as if the range class of the object property is annotated, e. g. `hasLocation` → {`hasLatitude, hasLongitude`}

Semantic Join. A *semantic join* creates an individual, representing a target parameter, by joining of multiple parameters of one source interface element. Assume, that a map publishes an event with parameters {`hasLatitude, hasLongitude`} and there is a point of interest finder that offers an operation consuming a parameter of type `Location`. Then, a semantic join is possible.

It has to be guaranteed, that the generated individual fulfills all constraints on OWL properties defined by the target class (and thus all superclasses).

Partial Substitution. Using a *partial substitution*, an OWL property of an individual represented as parameter can be updated with an individual or literal given by another parameter. With regard to Fig. 1, a partial substitution is possible between `Location` and `Event`, since the object of the OWL property `hasLocation` of an `Event` can be substituted by a `Location` individual. Partial substitutions are exclusively applicable for properties as target interface element. This is caused by the fact, that in our component model only properties expose and allow to change a partition of the component's data layer directly.

Using partial substitution increases the possibility to connect properties bidirectionally. As an example, consider that the calender has a property for its currently selected appointment and the map has a property for the currently selected location. Beside the possibility to semantically split the event to display its location, it is even feasible to connect the map to the calendar, to substitute the appointment's location with that of the map by dragging the map's marker.

Partial substitutions are not suitable for connections between events and operations for two reasons: First, it is not guaranteed that an event is correlated to an input interface element which can update the individual represented by the event. Second, events and operations in general hide the data layer.

Syntactic Join. A *syntactic join* is intended to synchronize `m` parameters published by `n` source interface elements of several *SC* and feeds them together in `1` target interface element. Several synchronization modes are supported.

- *tolerant*: The joiner waits until all sources have published at least once. Only the latest parameters are cached per source (the old value is overridden), and the cache is cleared after data transfer to the target.
- *repeating*: Here, the cache is not cleared, i. e., once all sources have sent data, each following publication causes the joiner to transfer data to the target.
- *queuing*: There is a queue per parameter. When all queues have at least one entry, the data is transferred to the target and the first element is removed.

3 Mediation-Equipped Platform for Mashups

3.1 Architecture

Our platform builds up on the CRUISe and EDYRA infrastructure we introduced earlier [1,2]. An overview is shown in Fig. 2. Universal composition is applied to create and execute presentation-oriented CWA, where components of the data, business logic and user interface (UI) layer share a generic component model. The latter characterizes components by means of several abstractions: parametrized events and operations, properties, and capabilities. The Semantic Mashup Component Description Language (SMCDL) serves as a declarative language implementing the component model. It features semantic annotations to clarify the meaning of component interfaces and capabilities [2]. Based on the component model, the declarative Mashup Composition Model (MCM) describes all aspects of a CWA, e. g. included components and event-based communication.

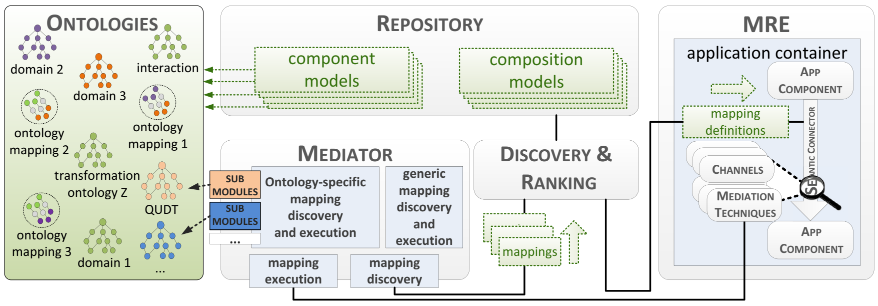


Fig. 2. Architectural overview of our mediation-equipped mashup platform

A *repository* is in charge of managing components and compositions. Furthermore it provides services for querying those artifacts.

Ontologies play an important role in our approach as they serve for annotating components and provide the schema knowledge most mediation techniques are based on. Besides domain-specific ontologies there can be (1) upper ontologies, e. g., for units, (2) ontologies defining how to transform concepts, and (3) mapping ontologies pointing out similarity relations of concepts in different knowledge representations. For the provision of mapping ontologies we assume that a state-of-the-art ontology alignment process takes place. Only confirmed mappings result in an ontology, linking concepts via predicates like `equivalentClass` and `sameAs`, e. g. `Location equivalentClass Localité`. A component can introduce new ontologies by using them for semantic annotation.

To enable recommendations, there are facilities for *discovery and ranking* of composition fragments, which represent composition knowledge, best matching user requirements and the current context. Within those modules, algorithms for recommendations utilize amongst others the semantic component annotations and ontology mappings. An essential task during discovery is the calculation of semantic connectors between components. Thereby, *mapping definitions* are derived utilizing the mediator or reused if already calculated.

A *mapping definition* specifies the mediation techniques required to align interface elements in a semantic connector. It is a data structure consisting of: an *ID*, P_{in} and P_{out} for more efficient matching and reuse, and one or more *mappings* including the composition and configuration mediation techniques, like the synchronization mode of a syntactic join.

A *mashup runtime environment (MRE)* interprets composition models in order to run mashups. With regard to data mediation, an MRE provides automatic support for the proposed mediation techniques, see Sect. 3.2 for details.

The *mediator* is responsible for mainly two tasks. First, it provides means for looking up mappings, which involve mediation techniques. To this end, the mediator takes two signatures, i. e., the URIs of concepts annotated at a source and a target interface element. In order to detect mappings e. g. between `milesDist` and `kilometerDist` in our scenario, algorithms have to inspect concepts on OWL property level, whereby we restrict the depth to one level. This task may result in multiple valid mappings, which have to be ranked further. Second, the mediator serves for the execution of mediation techniques defined in mapping definitions as requested by an MRE.

As illustrated in Fig. 2, there are generic and ontology-specific algorithms to accomplish these tasks. Generic algorithms utilize standard modeling constructs and reasoning rules of RDF/S and OWL. Mainly, relations like subsumption, property ranges and domains, `sameAs` and `equivalentClass` are leveraged.

The mediator can be extended with ontology-specific modules by implementing the required interface. They use dedicated modeling constructs and reasoning rules to derive and apply mappings and are consulted if generic algorithms cannot provide a mapping. Knowledge is encapsulated in modules for the algorithmic part and the corresponding ontologies for the terminology. Although in principle generically applicable, such modules are especially useful for conversions. There are modules per transformation ontology, responsible for interpreting and applying transformations on the payload delivered by events/properties. To identify suitable modules, each one provides a method to state if it supports the given signatures during discovery as well as execution of mappings.

Within the EDYRA project[1], we implemented a client-side thin-server MRE completely written in JavaScript. The mediator is distributed over the MRE and the SOAP-based *mediation service*. The latter is implemented in Java and uses the Jena framework for working with semantic models, including validating, reasoning and querying via SPARQL. The mapping discovery is located at the Java-based repository. We implemented the `DataSemanticsMatcher` as a generic algorithm that builds up on a QUDT-specific and a generic conversion module. The QUDT module supports annotated parameter types which refer to QUDT concepts. It queries the QUDT ontologies to check if both concepts are convertible, i. e., if they belong to the same unit domain. At execution time, the conversion multiplier is looked up and applied. The generic conversion module utilizes OWL constructs like `equivalentClass` and `equivalentProperty` to decide if two given classes are equal according to our definition in Sect. 2. To add new ontologies in our prototype, they have to be used for semantic annotation in SMCDL and registered manually at the repository and mediation service.

3.2 Runtime Support

Semantic connectors are implemented by associating the mapping definitions of semantic mediation techniques with a communication channel, and providing syntactic joins as built-in mediation components. The latter comply to the component model and can consequently be connected with other components. The MRE has templates for the SMCDL and the implementation of joiners and configures those as stated in the mapping definition to seamlessly instantiate and manage joiners like application components. With syntactic joins as components, single channels connect one SC with one TC.

[1] `http://mmt.inf.tu-dresden.de/edyra` (also links to our live demonstrator).

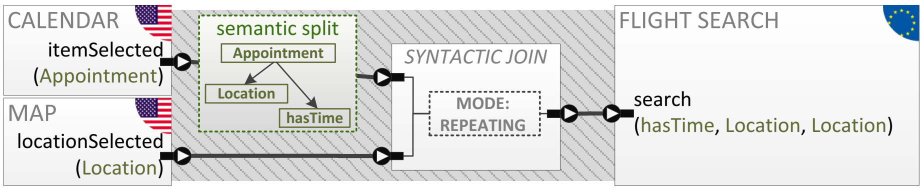


Fig. 3. Implementation of an exemplified semantic connector from the scenario

Fig. 3 shows an example (dashed area) of a semantic connector present in the mashup from our scenario. The semantic split is realized through a mapping definition which is directly attached to the channel between the calendar and a syntactic join. The latter aligns the split `Location` and `Date` as well as the `Location` from the map and transfers them combined to the flight search.

As described in our previous work [1], our components predominantly exchange data serialized in XML. There is a predefined *grounding* per ontology concept specifying the XML schema for individuals and literals. In case component developers utilize their own schemata or even other formats, like JSON, they have to ensure that data are transformed to the grounding, either as part of the component implementation or by transformation instructions in the SMCDL. In order to apply semantic mediation, data has to be available in RDF. Thus, we assume lifting and lowering transformations per predefined grounding.

When a SC publishes data according to the grounding, the channel delegates the execution of mediation techniques to the mediator by handing over the transported payload, the mapping definition, and in case of partial substitutions the target property's value. Per parameter, the mediator applies the lifting to get triples, which are then put in separate semantic models together with the terminological knowledge of ontologies. Using Jena in our prototype, validation and reasoning takes place automatically. Then the mediation techniques are executed as configured by the mapping definition. We utilize Jena's OWL API, e. g., to add OWL property values to individuals, and invoke conversion modules if required. Next, lowering takes place. Thereby, per target parameter, the model is queried with SPARQL, the results are serialized in the XML results format on which an XSLT transformation is applied. Finally, the mediated payload is forwarded to the TC. For simple datatypes as grounding, e. g. as input for conversions or result of a split, there is no dedicated lifting and lowering required. We programmatically map primitive data types to `Literals` and vice versa.

Syntactic joiners are connected to `n` inbound and one outbound channels. To achieve that, there are `n` operations in the joiner's generated SMCDL, whose parameter count and names correspond to those of the source interface element. Occurring event on an inbound channel are handled by the connected operation. The joiner extracts the parameter payload, handles it (e. g., adds it to a queue) and decides whether to fire its event according to its synchronization mode.

We conducted a benchmark where we measured the average response time of the mediation service's operations implementing the mediation techniques.

Table 1. Average response time of the mediation service in 100 runs per technique on a local server. The setup includes an Intel Core i7 2.8 GHz and 32 GB RAM.

Mediation technique	**Avg. response time**
Upcast (`AppointmentLocation` → `Location`)	≈15 ms
Conversion (`kilometers` → `miles`)	≈14 ms
Conversion (two equivalent classes)	≈40 ms
Semantic split (`Location` → {`hasLat, hasLng`})	≈23 ms
Semantic join (reversed split)	≈13 ms
Partial substitution (`Location` → `Event.hasLocation`)	≈23 ms

The results show a decent performance considering that lifting and lowering takes place in most cases. Network overhead of SOAP over HTTP may result in noticeable delays on slow connections, so that user experience may suffer in comparison to perfect match channels. That can be lowered by using WebSockets, and by integrating the mapping execution in a client-server MRE. Other crucial factors are ontology size and complexity, especially when a reasoner is attached.

4 Related Work

Proposals for semantically annotated services mostly use lifting and lowering to transfer data to the semantic layer [4,5]. While we use this, because our components do not exchange semantic data, our concept is not limited to upcasts.

Research shows that semantic web service descriptions are suited for mediation. There are different mediators in the conceptual framework of WSMO [6], which are provided as web services too and completely operate at a semantic layer. There are similar techniques involved, but due to the lack of a UI, execution performance is not that critical as for CWA. In addition, only a `1:1` communication is supported, while we can handle `n:m` semantic connectors.

For mashups, automated data mediation has been neglected so far [7]. Simple constructs exist that support, e. g., filtering, assignment and sorting of data, for example in Yahoo Pipes. But those rather belong to application logic than generic mediation techniques, and data semantics is not taken into consideration. Few approaches use semantically annotated component interfaces for matching at all, like [8,1]. Our previous work [1] can solve syntactical issues like different parameter naming with the help of wrappers. As a semantic issue only upcasts can be handled. Therefore, we largely extend this work.

5 Conclusion and Future Work

Since components of a CWA typically originate from different vendors, connecting them in meaningful way is challenging. Incompatibilities of signatures and exchanged data may, for instance, result from varying vocabularies or units. This complicates EUD, especially for non-programmers. In addition, connecting components in unforeseen ways becomes far from trivial. Thus, platforms for mashup EUD should feature means to automatically provide the required glue code.

We describe a set of data mediation techniques that use semantic component annotations to resolve interface mismatches. Those techniques reflect knowledge captured by ontologies, can be combined and are essential for establishing semantic connectors between components. The higher flexibility for combining components fosters re-usability and unforeseen coupling. This way, more niche requirements can be meet without the need for new components. Utilizing our implemented core platform, we show the practicability. However, due to the increased possibility of combinations, it is challenging to identify useful connectors. In addition, our approach depends on semantic annotations of OWL concepts, and we limit the depth to 1 when analyzing concepts during mapping discovery. Tough this may restrict the solution space, it lowers the algorithmic complexity.

Currently, we utilize mediation techniques for deriving and visualizing recommendations, in the CapView [2] and for synchronization of mediable components during collaboration. Future research will focus on the mediation of collections.

Acknowledgments. Carsten Radeck and Gregor Blichmann are funded by the German Federal Ministy of Economic Affairs and Energy (ref. no. 01MU13001D).

References

1. Pietschmann, S., Radeck, C., Meißner, K.: Semantics-based discovery, selection and mediation for presentation-oriented mashups. In: 5th Intl. Workshop on Web APIs and Service Mashups (Mashups), pp. 1–8. ACM (September 2011)
2. Radeck, C., Blichmann, G., Meißner, K.: Capview – functionality-aware visual mashup development for non-programmers. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 140–155. Springer, Heidelberg (2013)
3. Shvaiko, P., Euzenat, J.: Ontology Matching: State of the Art and Future Challenges. IEEE Transactions on Knowledge and Data Engineering 25(1), 158–176 (2013)
4. Szomszor, M., Payne, T., Moreau, L.: Automated syntactic medation for web service integration. In: Proc. of the Intl. Conf. on Web Services, pp. 127–136 (September 2006)
5. Nagarajan, M., Verma, K., Sheth, A.P., Miller, J.A.: Ontology driven data mediation in web services. Intl. Journal of Web Services Research 4(4), 104–126 (2007)
6. Mocan, A., Cimpian, E., Stollberg, M., Scharffe, F., Scicluna, J.: WSMO mediators (December 2005), `http://www.wsmo.org/TR/d29/`
7. Di Lorenzo, G., Hacid, H., Paik, H.Y., Benatallah, B.: Data integration in mashups. SIGMOD Rec. 38(1), 59–66 (2009), `http://doi.acm.org/10.1145/1558334.1558343`
8. Bianchini, D., De Antonellis, V., Melchiori, M.: A recommendation system for semantic mashup design. In: Workshop on Database and Expert Systems Applications (DEXA), pp. 159–163 (September 2010)

Standard-Based Integration of W3C and GeoSpatial Services: Quality Challenges

Michela Bertolotto[1], Pasquale Di Giovanni[1,2], Monica Sebillo[2], and Giuliana Vitiello[2]

[1] School of Computer Science and Informatics
Belfield, Dublin 4, Ireland
michela.bertolotto@ucd.ie
[2] Department of Management and Information Technology (DISTRA)
Via Giovanni Paolo II, 132-84084 Fisciano, SA, Italy
{pdigiovanni,msebillo,gvitiello}@unisa.it

Abstract. In recent years, Service Oriented Computing (SOC) has become one of the leading approaches for the design and implementation of distributed solutions. The key concepts are the notion of service and the possibility to seamlessly combine several modules to offer more sophisticated functionality. Such features were soon recognized by both W3C and OGC as relevant for their purposes, although their standards are incompatible and the seamless communication and exchange of information between these types of services are not directly achievable. The current most accepted solution to address this matter is represented by the development of a wrapper that manages technical issues that arise during the translation of requests and responses between them. However, the design of such a software module presents challenges in terms of infrastructure design and Quality of Service. In this paper we describe some issues to be faced when developing a service wrapper aimed at integrating existing geospatial services into a W3C service-based infrastructure.

1 Introduction

The Service Oriented Computing (SOC) paradigm has emerged as one of the leading approaches for designing and implementing distributed applications. The key idea behind this approach is the concept of service, an autonomous software module that, combined with other services, can be used to create complex solutions.

A service exposes its functionality through its public interface whose methods can be invoked by any software system without the need, for the service client, to know any detail about the service internal structure and business logic. However, since services and their clients can be developed by different entities, the first issues to address involve describing the public interface and providing a framework for data exchange in a wide accepted way and in a technology neutral manner.

In this context, the World Wide Web Consortium (W3C) has defined a series of universally accepted standards based on the use of the Extensible Markup Language (XML) in order to guarantee their independence from a specific platform or technology. The two most important W3C standards are the Web Services Description Language (WSDL) for the description of the service interface and the SOAP protocol for the exchange of messages [3], [13]. The flexibility and pervasiveness guaranteed by the W3C infrastructure has promoted during time the development of a growing number of service based solutions in many diverse fields [4].

The service-based approach has also become one of the preferred ways to discover, access, and manage geographic information. The ability to offer traditional Geographic Information Systems (GIS) capabilities in a distributed manner has been recognized by the geographic community as a valuable opportunity to provide new ways to use geospatial information and increase its distribution. However, despite the wide acceptance of the W3C proposals as a means to promote interoperability and platform independence, the GIS community has developed, over time, its own set of standards for the fulfilment of geospatial data oriented services. In particular, the proposals of the Open Geospatial Consortium (OGC), which represents the reference organization for "the development of international standards for geospatial interoperability" (http://www.opengeospatial.org/), have become the *de facto* standard for developing distributed geographic applications.

Unfortunately, although both W3C and OGC standards are based on XML for data exchange and HTTP as the transport protocol, some design choices make them totally incompatible. Nevertheless, a better integration between these two worlds could be of interest for both communities. In fact, the former could access and process the wide amount of geospatial data currently available only by invoking OGC services, while the latter could benefit both from additional standards, such as those for access management and security, and from the huge amount of supporting infrastructures for W3C services. Such awareness is stimulating a radical revision of such standards to remove their intrinsic incompatibilities.

The currently most accepted solution to this aim is represented by the development of a software wrapper, usually a service itself, that "translates" the requests and responses messages from the W3C services format to a format suitable for the OGC services and vice-versa, while keeping the structure of the original services unchanged [2], [5], [15]. However, the concrete development of a wrapper does not represent the only issue to solve since, independently of the actual set of adopted standards, every service-oriented solution cannot disregard fundamental non-functional requirements, such as the quality of the provided information, its security, the service response time. In fact, taking into account essential Quality of Service (QoS) aspects is of utmost importance to guarantee a satisfactory computation and make the SOC paradigm a feasible option for the development of complex distributed solutions.

In this paper we discuss our research in the area of geospatial service-oriented architectures. We analyze the challenges involved in the integration and reuse of heterogeneous services with focus on QoS aspects, and propose recommendations for the development of a viable solution that takes such aspects into account.

The remainder of this paper is organized as follows. In Section 2 we provide an overview of the two main W3C standards and compare them with the OGC proposals. In Section 3 we briefly describe the most important QoS attributes that directly impact on the development of service-based solutions, and discuss the QoS issues that affect the development of distributed solutions for geospatial data. In Section 4, we describe the main challenges that arise during the design of a wrapper addressed to the W3C-OGC services dialogue. Some conclusions are drawn in Section 5.

2 W3C and OGC Standards for Service Based Development

The functionality of a service is exposed through its public interface and the communication between a service and its clients is based on various messages exchange patterns. However, the actual definition of the public interface and the structure of messages are strongly related to the particular set of standards adopted. In this section we provide an overview of the main characteristics of the two major W3C standards, namely WSDL and the SOAP protocol, and compare them with the three principal OGC proposals, namely the Web Map Service (WMS), the Web Feature Service (WFS) and the Web Coverage Service (WCS) standards.

WSDL [3] is an XML based language for describing W3C services. A WSDL document separates the abstract aspects of a service description from more concrete ones, such as the binding to a certain network protocol. The typical structure of a WSDL document is made up of seven elements, namely Types, Message, Operation, Port Type, Binding, Port and Service. In particular, the former four are meant to statically define the public interface of a Web service, while the latter three are used to bind the interface to a concrete network protocol. A direct drawback of such design choices is that, in a W3C-oriented environment, the structure of the message payload has to be completely specified at design time [22].

SOAP is "a lightweight protocol for exchange of information in a decentralized, distributed environment” [13]. Three basic components characterize a typical SOAP-based message: an Envelope, a Header and a Body. The Envelope can be seen as the container of the message itself. The optional Header field can be used to carry additional information useful to guarantee some properties, such as security and reliability of exchanged messages. The Body element encompasses the real payload of the exchanged message.

When compared to the W3C choices, the design philosophy of the OGC standards is quite different, and the main dissimilarities between them concern the different approach for the public interface design, and the binding type and the binding time of operations. Two further clear differences are the basic role of the Geography Markup Language (GML) [21] and the fact that each type of OGC service is based on a separate standard explicitly designed to deal with a specific kind of data. Moreover, they also specify the functionality offered by the service interface along with the possibly needed additional data structures. A direct consequence of this choice is that, with OGC services, the actual structure of the message payload can be known only at run time, differently from what occurs in W3C environments.

As for the service specification, the most widespread and commonly used are WMS, WFS and WCS. A WMS allows clients to request georeferenced map images from one or more geospatial databases. WFS allows for accessing and manipulating geographic features. Finally, the WCS defines an interface for the exchange of geospatial information representing phenomena that can vary in space and time, known as coverages. The only functionality that is common to these three types of services is GetCapabilities, it allows a geospatial service to expose its capabilities to clients.

3 QoS Issues in Geospatial Web Services

The increasing adoption of the OGC proposals as a concrete means to access and make use of geospatial data in a distributed and vendor independent manner, has shifted the attention from data and information supply to information quality and implementation of services themselves. Thus, also for OGC services, the assessment of the most common QoS attributes is becoming fundamental to distinguish between reliable and non-reliable services. Generally speaking, QoS within the SOC paradigm represents an important and widely discussed topic, due to its basic role in various key aspects. The scientific and industrial communities have defined several main QoS categories and various attributes for each of them that contribute to the fulfillment of the desired QoS property. Moreover, a W3C working group [19] has identified a set of basic QoS requirements that have to be taken into account during the development of a Web service, namely Performance, Reliability, Scalability, Capacity, Robustness, Exception Handling, Accuracy, Integrity, Accessibility, Availability, Interoperability and Security. A complete analysis of these can be found in [20], while approaches to express and describe QoS characteristics and metrics can be found in [18].

As for the quality of geospatial Web services, basic assumptions about QoS attributes still hold. However, their evaluation must be performed according to both the specific characteristics of geospatial data and the way it is handled by OGC-compliant solutions. In addition, another relevant aspect that must be taken into account is represented by the technical differences among the various software implementations of the OGC standards and the related supporting infrastructure.

From a high level perspective, the process of obtaining knowledge from geospatial information can be viewed as a three step process, namely querying the data, assembling the retrieved subset and finally performing the effective computation [25]. The first issue along this sequence of operations is represented by the specific characteristics of geospatial data that usually is voluminous and heterogeneous, distributed among different data silos and can suffer from access restrictions due to institutional policies [11]. Such characteristics have, of course, a significant impact on the actual quality of the final information offered by geospatial services to third party users. In fact, as clearly discussed in [12], due to the common practice of combining data from multiple sources, geospatial datasets are inclined to contain errors since the various providers can make, for example, different assumptions about data structure. As defined in [12], the most important quality components for geospatial data are lineage, completeness, logical consistency, attribute accuracy and positional accuracy.

The aforementioned quality attributes are useful to assess also the quality of metadata that, due to its importance in this context, must be accurately evaluated. Indeed, a poor quality metadata determines the lack of information quality and can lead final users to formulate wrong assumptions about the received dataset.

The ISO19113 standard, instead, identifies five criteria for geospatial data quality, namely positional accuracy, temporal accuracy, logical accuracy, thematic accuracy, and completeness [17]. Finally, a recent factor that influences the quality of geospatial data is represented by the creation of user-generated geospatial content and Web 2.0. How to efficiently assess the quality of such a type of data is still an open research question. An example of filtering and composition of Web 2.0 sources can be found in [1].

As for the quality factors that mainly impact on the actual development of geospatial services dealing with significant amount of data, a first important discussion can be found in [10]. This document shows how, from a general point of view, the quality attributes proposed by W3C and mentioned in the previous section can be applied to geospatial services, except for the scalability requirement. In [9] some more specific directives and obligations for implemented services are mentioned. In particular, the three fundamental QoS criteria to respect are:

- performance: the time for sending the initial response to a discovery service request shall be maximum 3 seconds in normal situations. Normal situations represent out of peak load periods, i.e., 90 % of the time;
- capacity: the minimum number of simultaneous requests served by a discovery service according to the performance quality of service shall be 30 per second;
- availability: the probability of a network service to be available shall be 99% of the time.

In [24] the common issues impacting on the overall QoS and concerning current proposals and implementations of OGC standards are discussed. The authors divide those issues into three levels, namely standard definition, software implementation of the standard, and software application. Among the various problems, the following are functional to the goal of the present discussion: the lack of a standardized authorization/authentication mechanism, the misuse of the standardized HTTP error codes, the version proliferation, the discrimination between mandatory and optional features, and the high level of autonomy offered by the various standard specifications.

A concrete example of QoS issues in a real software solution can be found in [26]. In the development of their prototype for real time geospatial data sharing over the Web, authors notice how the adoption of OGC standards is useful to solve problems at the syntactic level, while several issues may arise at the semantic level. System reliability represents the second important problem that is particularly accentuated when OGC services are provided by different entities. Security is another major concern. Finally, performance bottlenecks due to the transfer of redundant XML data over the network and the high cost of the parsing XML messages have a serious impact on the effective use of the proposed solution.

In [11] several OGC-compliant services implementations are tested. The results related to relevant performance parameters, show how, due to the GML verbose nature, a consistent number of bottlenecks may arise when there is the need to transfer large amount of geospatial data. Moreover, different software solutions vary in the way OGC specifications are implemented. Two direct consequences may arise from such dissimilarities, namely the reduced quality that can be perceived by final users and critical interoperability problems.

4 A Wrapper-Based Solution

In order to face effects deriving from technical and semantic differences between W3C and OGC services the currently most accepted solution is represented by a software wrapper that manages most of the technical topics that arise during the translation of requests and responses [15]. However, such a translation cannot be automated due to several issues that need to be carefully taken into account during the wrapper design to make this solution a feasible option. First of all, a wrapper is usually a service itself, then it requires a typical supporting infrastructure of service-based solutions, while its design might be influenced by the specific needs of the application under development. Indeed, two symmetrical types of wrapper can be developed, either adapting the interface of an OGC service to the technical requirements of a W3C-based infrastructure or vice-versa. Existing W3C services providing geospatial information that could be useful in an OGC-based Spatial Data Infrastructure (SDI) constitute an example of the latter case. The second cause of difficulties is represented by the number of services whose functionality has to be exposed by the intended wrapper. In fact, although the simplest solution concerns a one-to-one mapping, i.e., a wrapper adapts the interface and functionality of a single W3C / OGC service, it is also possible for it to gather functionality of different services. A typical example is constituted by a W3C service that offers, in a single WSDL document, the methods to access the data layers of either two WFSs or a WFS and a WMS. Finally, a further issue concerns the need to properly structure the WSDL document in order to distinguish among the various OGC services since the public interface and signatures of the implemented methods are rigorously standardized by the Consortium.

In the following, we discuss some challenges about the design of a one-to-one wrapper by describing a concrete example of an OGC to W3C mapping, Moreover, some basic QoS parameters are investigated that are affected when offering geospatial data coming from other OGC services and exploited through W3C standards.

As a concrete example where an OGC-to-W3C wrapper can actually promote and support a better information exchange between different entities, we illustrate its usage in the context of a research activity aimed at helping Sri Lankan farmers improve their productivity by providing them with customized and up-to-date information, such as the current selling prices of a product. Such an activity constitutes a pilot study for the Social Life Networks for the Middle of the Pyramid (SLN4MoP) project, an international collaborative research program that aims at providing real-time information to meet the daily needs of people living in developing countries [23].

The proposed system is based on a client-server architecture, although some technological constraints and the elicited needs of the involved stakeholders deeply influenced its overall design. As for the client tier, a common trait in many developing countries is the wide spread of mobile devices compared to the diffusion of traditional PCs. Such a factor led us to propose a mobile solution for the actual application with which the farmers interact. Detailed information about the implications and design challenges of our choice can be found in [6,7].

As for the back-end, the blueprint of the architecture has been organized by exploiting the principles of the SOC paradigm, which better comply with the *in progress* nature of SLN4MoP project, that is, providing its functionality as set of interacting services helped us to easily satisfy several fundamental design goals and QoS parameters. In particular, we needed both a reliable and flexible infrastructure, where new software modules can be added and can communicate with the existing ones without affecting the original design and behavior, and a reduced complexity during the access to heterogeneous and distributed data sources hiding, at the same time, the underlying different storage formats.

As for the QoS aspects, since the business processes are now decomposed into a series of interacting services, the availability, interoperability and performance parameters are of utmost importance for an efficient usage of this system. However, while availability strongly depends on the failure ratio of the underlying supporting components, interoperability and performance deserve further considerations.

To support our discussion, we consider the following real scenario. A governmental officer needs to visualize on a map the position of all local markets of a given district along with the selling prices of certain crops. The required operation corresponds to a combination of two atomic functions, namely the provision of various data units for the composition of a map, and a list of scalar values. The former is a typical functionality offered by an OGC service, the latter can be provided by a W3C service. In order to derive the expected result, it is necessary to invoke an advanced service capable to split and direct the atomic requests towards components in charge of performing them, and then combine responses deriving from them as a unique output. This capability represents a fundamental feature of the SOC paradigm: the services composition, namely the ability to compose services to obtain complex results.

One of the most common types of composition is service orchestration where the messages exchanged among services and the execution order of their interactions, is coordinated by a central controller. In order to effectively make the orchestration possible, all the involved services need to share the same Interface Description Language (IDL) and the same framework for the messages exchange. Then, the interoperability in the context of services orchestration represents a key requirement, but, as shown in our scenario, protocols based on different rules for the definition of the public interface and the message exchange system, make services orchestration not directly achievable.

The solution we have proposed is based on a wrapper addressed to a syntactic translation from OGC to W3C, which exploits existing orchestration middleware and the well-established services orchestration in W3C environments. In particular, the task performed by the proposed wrapper consists in the translation of SOAP-based messages into OGC-compliant requests and vice-versa. Such a task can be partitioned into four main steps:

1. the wrapper receives, from a W3C service a SOAP message containing a request for a specific geospatial dataset;
2. the wrapper translates the SOAP-based request into a format suitable for the underlying OGC service, and sends the query;
3. the OGC service returns the desired information;
4. the wrapper translates the received response into a SOAP-compliant format and sends it back to the requesting client.

However, a wrapper-based solution presents an important drawback, namely a serious impact on the overall composition performance. Such an aspect, in the context of our project, cannot be underestimated and requires further investigation.

Besides traditional aspects (such as, the quality of the underlying network that contributes to the achievement of a satisfactory performance level), a relevant factor for performances is represented by the specific characteristics of geospatial information (described in Section 3). As compared to the size of traditional SOAP messages, the size of geospatial data is usually several orders of magnitude larger. Since in the wrapper-based solution such data has to be packaged in the Body element of a SOAP message, it is clear that encoding, decoding and transmission of SOAP messages represent new significant issues. Such a problem has a direct impact on measurable values (like response time or throughput) directly related to the Quality of Experience of final users. Moreover, it might also influence the behavior of other aspects of the entire Service Oriented Architecture (SOA) to which the wrapper belongs, such as the transaction management protocols and the above described services orchestration. In particular, in a traditional orchestration the execution of an operation may depend on the output of a previous computation, and data complexity. The above described scenario deals with high volumes of data and long running operations, then the considerable amount of waiting time needed to process or simply transfer SOAP-encoded geographic data may cause a throughput reduction. In the worst case, a time-out error may occur that causes the entire workflow blocking. An asynchronous strategy based on appropriate SOAP message patterns (e.g., Fire and Forget) [8] represents a possible solution for all wrapper-based and time-consuming tasks.

Another aspect that adversely affects the performance and effectiveness of a wrapper is related to the supplementary delay caused by the need to query remote OGC sources. Information caching represents a feasible solution to reduce this inconvenience and improve performance and overall scalability. Some considerations about the design choices of OGC services support this option. In particular, most of OGC services are basically read-only services whose queries "access groups of features rather than individual features" [16]. Of course, traditional cache invalidation mechanisms (on demand, time limited, etc.) can be used to force the wrapper to invoke the original data source and refresh the local cache. Examples of cacheable items are the Capabilities document returned by the invocation of the GetCapabilities function, and the GML Schemas returned by the DescribeFeatureType function of a WFS.

A further service property to be taken into account when designing a wrapper, concerns its level of flexibility and reusability (a desirable property in the SOC paradigm). Such parameters are related to the granularity of a service, namely its size. In [14], the authors classify service granularity into three different categories: functionality granularity, data granularity and business value granularity. In a wrapper-based solution for service orchestration, data granularity represents the unique parameter that can be investigated during the design. Some optimizations can be done, however such parameters depend on the implementation and specific choices made for the original OGC service. A detailed discussion about this topic can be found in [16].

5 Conclusions

The goal of the research we are conducting is to define an infrastructure for the provision of heterogeneous Web services within a geographic information system. In particular, the focus of our current efforts is on the orchestration of traditional and geospatial services. The solution we have proposed is based on a wrapper that integrates W3C and OGC services in a seamlessly manner. In this paper we have discussed the QoS parameters that should be properly considered in this context. We have emphasized that, besides parameters that the literature suggests to take into account when dealing with these two standards separately, it is necessary to include some criteria that exclusively derive from the growing complexity of the integrated solution, such as supplementary delay and throughput reduction. Indeed, a wrapper-based solution implies a notable impact on the service performances and effectiveness, and then it is essential to handle those QoS parameters during the design phase in order to perform the best choices, independently from the technology used in the subsequent implementation step. In the future, we plan to complete the infrastructure proposed for SLN4MoP, and stress it by testing its performances against a large amount of data.

References

1. Barbagallo, D., Cappiello, C., Francalanci, C., Matera, M., Picozzi, M.: Informing observers: quality-driven filtering and composition of web 2.0 sources. In: EDBT/ICDT Workshops, pp. 1–8 (2012)
2. Bertolotto, M., Di Giovanni, P., Sebillo, M., Tortora, G., Vitiello, G.: The Information Technology in Support of Everyday Activities: Challenges and Opportunities of the Service Oriented Computing. Mondo Digitale (2014)
3. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1. World Wide Web Consortium (2001), `http://www.w3.org/TR/wsdl`
4. Costagliola, G., Casella, G., Ferrucci, F., Polese, G., Scanniello, G.: A SCORM Thin Client Architecture for e-learning Systems Based on Web Services. Int. J. of Distance Education Technologies. 5(1), 19–36 (2007)
5. Di Giovanni, P., Bertolotto, M., Vitiello, G., Sebillo, M.: Web Services Composition and Geographic Information. In: Pourabbas, E. (ed.) Geographical Information Systems: Trends and Technologies, pp. 104–141. CRC Press (to appear, 2014)
6. Di Giovanni, P., Romano, M., Sebillo, M., Tortora, G., Vitiello, G., Ginige, T., De Silva, L., Goonethilaka, J., Wikramanayake, G., Ginige, A.: User Centered Scenario Based Approach for Developing Mobile Interfaces for Social Life Networks. In: First International Workshop on Usability and Accessibility Focused Requirements Engineering (UsARE 2012), pp. 18–24. IEEE (2012)
7. Di Giovanni, P., Romano, M., Sebillo, M., Tortora, G., Vitiello, G., Ginige, T., De Silva, L., Goonethilaka, J., Wikramanayake, G., Ginige, A.: Building Social Life Networks Through Mobile Interfaces: The Case Study of Sri Lanka Farmers. In: Spagnoletti, P. (ed.) Organizational Change and Information Systems. LNISO, vol. 2, pp. 399–408. Springer, Heidelberg (2013)
8. Erl, T.: Service-oriented architecture: concepts, technology, and design. Prentice Hall, PTR (2005)

9. European Commission: Commission Regulation 1088/2010 amending Regulation (EC) No 976/2009 as regards download services and transformation services (2010), `http:seur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:02009R0976-20101228:EN:NOT`
10. European Commission: INSPIRE Network Services Performance Guidelines (2007)
11. Giuliani, G., Dubois, A., Lacroix, P.: Testing OGC Web Feature and Coverage Service performance: Towards efficient delivery of geospatial data. J. of Spatial Information Science. 7, 1–23 (2013)
12. Goodchild, M.F., Clarke, K.C.: Data Quality in Massive Data Sets. In: Abello, J., Pardalos, P., Resende, M.G.C. (eds.) Handbook of Massive Data Sets, pp. 643–659. Kluwer Academic Publishers, The Netherlands (2002)
13. Gudgin, M., Hadley, M., Moreau, J.J., Nielsen, H.F.: SOAP Version 1.2. World Wide Web Consortium (2001), `http://www.w3.org/TR/2001/WD-soap12-20010709/`
14. Haesen, R., Snoeck, M., Lemahieu, W., Poelmans, S.: On the Definition of Service Granularity and Its Architectural Impact. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 375–389. Springer, Heidelberg (2008)
15. Ioup, E., Lin, B., Sample, J., Shaw, K., Rabemanantsoa, A., Reimbold, J.: Geospatial Web Services: Bridging the Gap between OGC and Web Services. In: Sample, J.T., Shaw, K., Tu, S., Abdelguerfi, M. (eds.) Geospatial Services and Applications for the Internet, pp. 73–93. Springer, New York (2008)
16. Ioup, E., Sample, J.: Managing Granularity in Design and Implementation of Geospatial Web Services. In: Zhao, P., Di, L. (eds.) Geospatial Web Services: Advances in Information Interoperability, pp. 18–35. IGI Global, New York (2011)
17. ISO 19113: Geographic information - Quality principles. International Organization for Standardization (2002)
18. Kritikos, K., Pernici, B., Plebani, P., Cappiello, C., Comuzzi, M., Benrernou, S., Brandic, I., Kertész, A., Parkin, M., Carro, M.: A Survey on Service Quality Description. ACM Computing Surveys 46(1), Article 1 (2013)
19. Lee, K.G., Jeon, J.H., Lee, W.S., Jeong, S.-H., Park, S.-W.: QoS for Web Services: Requirements and Possible Approaches. World Wide Web Consortium (2003)
20. O'Brien, L., Bass, L., Merson, P.F.: Quality Attributes and Service-Oriented Architectures. Software Engineering Institute (2005)
21. Portele, C.: OpenGIS Geography Markup Language (GML) Encoding Standard. Open Geospatial Consortium Inc. (2007), `http://www.opengeospatial.org/standards/gml`
22. Schäffer, B.: OWS 5 SOAP/WSDL Common Engineering Report. Open Geospatial Consortium Inc. (2008), `http://www.opengeospatial.org/standards/dp`
23. Sebillo, M., Tortora, G., Vitiello, G., Di Giovanni, P., Romano, M.: A Framework for Community-Oriented Mobile Interaction Design in Emerging Regions. In: Kurosu, M. (ed.) Human-Computer Interaction, HCII 2013, Part III. LNCS, vol. 8006, pp. 342–351. Springer, Heidelberg (2013)
24. Vanmeulebrouk, B., Bulens, J., Krause, A., de Groot, H.: OGC standards in daily practice: gaps and difficulties found in their use. In: GSDI11 World Conference (2009)
25. Wei, Y., Santhana-Vannan, S.-K., Cook, R.B.: Discover, Visualize, and Deliver Geospatial Data through OGC Standards-based WebGIS System. In: 17th International Conference on Geoinformatics, pp. 1–6. IEEE (2009)
26. Zhang, C., Li, W.: The Roles of Web Feature and Web Map Services in Real-time Geospatial Data Sharing for Time-critical Applications. Cartography and Geographic Information Science 32(4), 269–283 (2005)

Tamper-Evident User Profiles for WebID-Based Social Networks

Stefan Wild, Falko Braune, Dominik Pretzsch, Michel Rienäcker, and Martin Gaedke

Technische Universität Chemnitz, Germany
{firstname.lastname}@informatik.tu-chemnitz.de

Abstract. Empowering people to express themselves in global communities, social networks became almost indispensable for exchanging user-generated content. User profiles are essential elements of social networks. They represent their members, but also disclose personal data to companies. W3C's WebID offers an alternative to centralized social networks that aims at providing control about personal data. WebID relies on trusting the systems that host user profiles. There is a risk that attackers exploit this trust by tampering user profile data or stealing identities. In this paper, we therefore propose the IronClad approach. It improves trustworthiness by introducing tamper-evident WebID profiles. IronClad takes protective measures to publicly discover malicious manipulation of profile data. We exemplarily implement IronClad in an existing WebID identity management platform known from previous work.

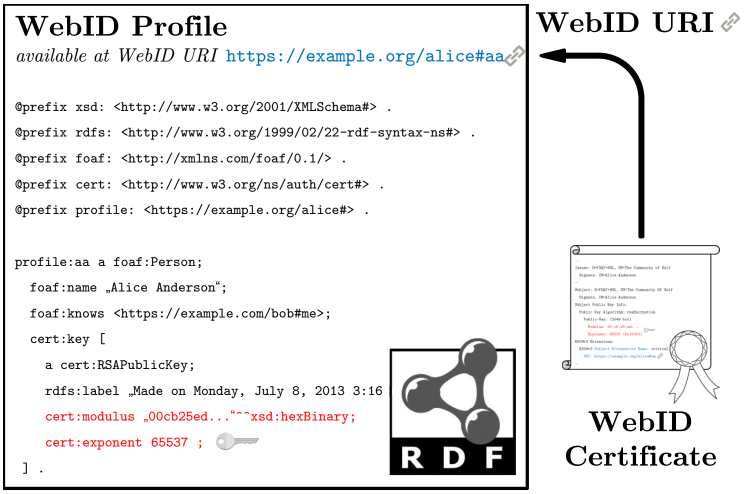
Keywords: Identity, WebID, Linked Data, Social Networks, Trust, Security, Data Integrity, Protection, Tamper Detection.

1 Introduction

Social networks have become crucial elements in modern society. They allow people to connect, communicate, and express themselves on a global scale. Joining social networks usually requires creating user profiles. Users therefore need to entrust personal information to network providers which escrow the information for them. This enables identification, eases discovery, and digitally represents the user. However, it also creates another copy of the user's digital identity. In today's ecosystem of social networks each copy needs to be maintained separately. This makes exchange and update of personal data across different domains and organizational boundaries difficult. Thus, centralized social networks like Twitter and Facebook gave rise to customer lock-in for millions of users [15].

For avoiding such walled gardens, the W3C devised WebID. As an open, universal, and decentralized identification approach, WebID allows users to be their own identity provider and establish their own personal social networks [11]. With WebID, users are enabled to manage their personal information at a self-defined place. They can also employ their WebID identities for global authentication.

WebID consists of three interrelated artifacts, which are illustrated in Figure 1: The *WebID URI* is a unique identifier referring to an agent. While an agent is typically a person, it can also be a robot, group or any other entity that needs to be identified. The *WebID certificate* is a common X.509v3 client certificate. It includes a public key and a WebID URI linking to the WebID profile. The *WebID profile* is a resource containing personal information about the identity owner. Personal information are described in an extensible and machine-readable way using Linked Data. Each WebID profile also stores public keys. They are used along with the corresponding WebID certificate for an ownership-based authentication defined by the WebID protocol.



```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix cert: <http://www.w3.org/ns/auth/cert#> .
@prefix profile: <https://example.org/alice#> .

profile:aa a foaf:Person;
  foaf:name „Alice Anderson“;
  foaf:knows <https://example.com/bob#me>;
  cert:key [
    a cert:RSAPublicKey;
    rdfs:label „Made on Monday, July 8, 2013 3:16
    cert:modulus „00cb25ed...“^^xsd:hexBinary;
    cert:exponent 65537 ;
 ] .
```

```
...
Issuer: O=FOAF+SSL, OU=The Community of Self
  Signers, CN=Alice Anderson
...
Subject: O=FOAF+SSL, OU=The Community Of Self
  Signers, CN=Alice Anderson
Subject Public Key Info:
  Public Key Algorithm: rsaEncryption
    Public-Key: (2048 bit)
      Modulus: 00:cb:25:ed: ...
      Exponent: 65537 (0x10001)
X509v3 Extensions:
  X509v3 Subject Alternative Name: critical
    URI: https://example.org/alice#aa
...
```

Fig. 1. Artifacts in WebID

Problem. Despite all advantages, being an identification mechanism that does not rely on authorities makes WebID vulnerable to attacks on user identities. WebID allows users to host their profiles at arbitrary locations. Yet it does neither ensure nor verify data integrity of user profiles. Experienced users know how to set up and protect a system storing their WebID profile. Inexperienced users, however, do not know this and would probably prefer using third-party managed services for hosting. Consequently, inexperienced users must trust third parties to not accessing and tampering their profile data [6]. The trustworthiness of managed services can hardly be assessed or guaranteed [5]. This shifts the problem from trusting identity providers [4] to trusting cloud storage providers. Requestors therefore depend on external means to decide whether to trust profile data. Obtaining write access to a WebID profile would enable an attacker to tamper user data stored inside [5]. Tampering user profile data, e.g., changing the e-mail address or replacing social contacts, could interfere with further transaction. Having a chance to add a public key to the identity owner's WebID profile would allow constructing a client certificate with the corresponding private key. Attackers could then use such certificate for authenticating to services as the identity owner without her knowledge and intent.

Objective. To increase trustworthiness in WebID-based social networks, we aim at providing a means for users to publicly detect tampering of profile data. For achieving this objective, we propose the novel IronClad approach for tamper-evident[1] WebID profiles. We therefore provide contributions for:

– Signing WebID profile data,
– Discovering WebID identity theft, and
– Verifying WebID profile data integrity.

Impact. Achieving the objective would allow for storing WebID profiles in potentially harmful environments, reducing the entry barrier for inexperienced users and, thus, contributing to increase the overall security and adoption of the WebID identity mechanism. Otherwise, users still risk to retrieve WebID profile data that is not in accordance with the identity owner's original intention.

The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 describes and exemplarily demonstrates the IronClad approach for providing tamper-evident WebID profiles. Section 4 evaluates the approach. Section 5 concludes the paper.

2 Related Work

Literature to file systems and database systems broadly deals with the topic of ensuring data integrity. This discussion of related work focuses on the applicability of integrating features for detecting tampering attacks to Web-based systems only. To structure the discussion, we pay particular attention to interoperability, applicability and accessibility of tamper-evident features.

Centralized social networking platforms like Facebook and Google+ enable users to create views on their profile data. Such views can conceal sensitive data to external parties, e.g., groups of requestors. When targeting profile data disclosure or malicious manipulation, they are, however, inapplicable to detect internal read/write attacks without further ado, as described by Feldman et al. in [6]. Through establishing decentralized social networks, WebID distributes this problem to systems that host the profiles. Contrary to centralized social networking platforms, Web Access Control[2] (WAC) facilitates securing resources in a decoupled and decentralized way. However, WAC also focuses on access protection and not on data protection [14]. Even though protection of personal data could be accomplished through encryption, this is inappropriate in WebID. Profiles have to be, at least partially, accessible for authentication of identity owners and for queries of requestors. It would be also required to either distribute keys for decryption to an unknown number of potential requestors or establish central authorities for key management, which does not conform to WebID's idea of focusing on individuals.

[1] "Tamper-evident" is commonly defined as a means for making unauthorized access to a protected object easily detectable.
[2] `http://www.w3.org/wiki/WebAccessControl`

With regard to detect tampering of profile data, WebID shares similar disadvantages with other identity management systems like OpenID or Mozilla Persona [8]. OpenID implements only limited handling of personal attributes [7], whereas Persona is not designed for attaching profile data to an identity in a holistic way [1]. In contrast, WebID allows flexibly extending profile data, i.e., add cryptographic signatures. Such extension is not applicable to many social networking platforms and identification systems due to their centralized, closed or restricted handling of user profile data.

Public keys and signatures must also be protected from manipulation in order to provide sound proof of the identity owner's intent. This could be accomplished by a public key infrastructure (PKI) involving certificate authorities (CA) or a Web of Trust (WoT). A PKI based on CAs represents a centralized trust model that uses hierarchically organized authority chains [2]. WebID allows for adapting this model, e.g., similar to signing WebID certificates by a trusted third party instead of the identity owner himself[3]. However, we do not want to impair WebID's decentralized approach of involving and empowering individuals instead of authorities. By contrast, the WoT concept represents a flat hierarchy only relying on individuals [2]. It needs member discovery and makes updating public keys and signatures difficult due to their necessary distribution and inclusion in other data stores, e.g., user profiles.

3 Tamper-Evident WebID Profiles through IronClad

In order to detect tampering of WebID profile data, we created IronClad. It is based on the principle that only identity owners should be enabled to change their profile data in a sustainable way. For ensuring requestors that all WebID profile data is what was intended by the identity owner, IronClad incorporates three main activities: signing profile data, storing/retrieving signatures, and verifying data integrity of profiles. They are according to our key contributions. We illustrated them using BPMN in Figure 2 and describe them in the following.

3.1 Operations Supported by IronClad

Signing WebID Profile Data. By signing the WebID profile, the identity owner (cf. top of Figure 2) proves that personal data stored in his profile is sound and was not changed by another party. In order to avoid signing tampered data, the data integrity of the WebID profile needs to be checked (cf. ① in Figure 2) prior to updating relevant data (cf. ②) and creating signatures (cf. ③). Algorithm 1 specifies in pseudo code notation how IronClad creates signatures of WebID profile data. Our approach uses the RDF graph representation of a WebID profile for computing hash values independent from specific data serializations, e.g., RDF/XML or Turtle. To address different orders of RDF

[3] In WebID, certificates are usually self-signed as the trust does not rely on the public key, but on the WebID URI linked resource storing the public key, which we want to protect against tampering.

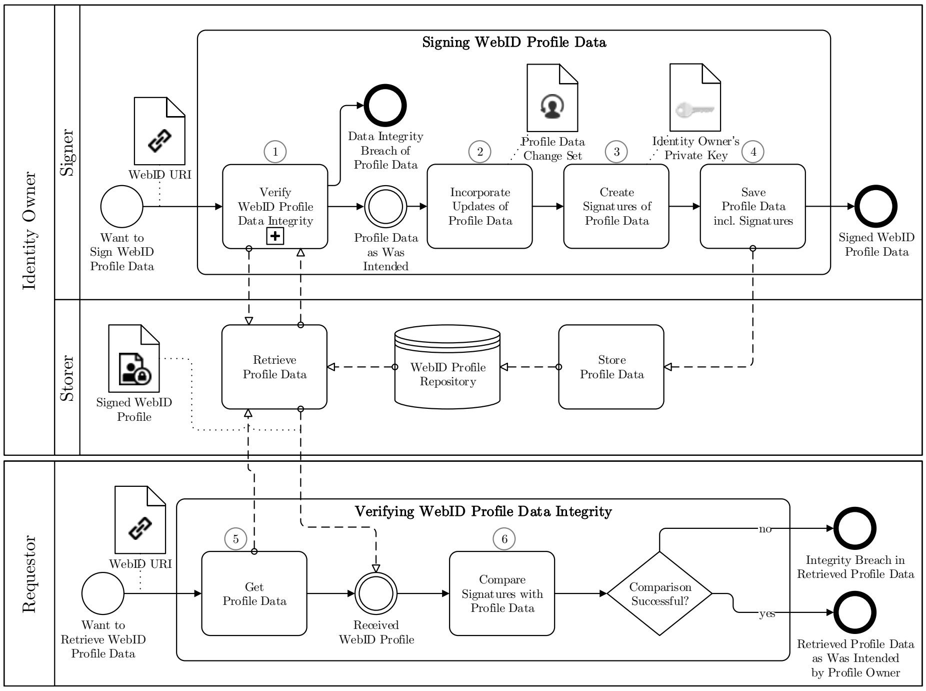

Identity Owner
Signer
Storer
Requestor
Signing WebID Profile Data
WebID URI
Want to Sign WebID Profile Data
1
Verify WebID Profile Data Integrity
Data Integrity Breach of Profile Data
Profile Data as Was Intended
2
Incorporate Updates of Profile Data
Profile Data Change Set
3
Create Signatures of Profile Data
Identity Owner's Private Key
4
Save Profile Data incl. Signatures
Signed WebID Profile Data
Signed WebID Profile
Retrieve Profile Data
WebID Profile Repository
Store Profile Data
Verifying WebID Profile Data Integrity
WebID URI
Want to Retrieve WebID Profile Data
5
Get Profile Data
Received WebID Profile
6
Compare Signatures with Profile Data
Comparison Successful?
no
yes
Integrity Breach in Retrieved Profile Data
Retrieved Profile Data as Was Intended by Profile Owner

Fig. 2. Big picture of IronClad: Signing, storing, and verifying data integrity of profiles

triples and blank nodes, IronClad performs a canonicalization. We therefore utilize the One-Step Deterministic Labeling method proposed by Carroll in [3] and the methodology described by Tummarello et al. in [12].

To avoid disclosing the identity owner's private key to a third party, the signing process is split into a server and a client side part. IronClad's server side computes hash values of each *minimal self-contained graph* [12] found in the graph representation of the WebID profile. It combines all hash values to a signing request afterwards (cf. lines 4 to 8). The client side part analyzes this request and signs the content. It creates the signature through encrypting each hash value with a private key (cf. lines 9 to 12). The identity owner selects the corresponding private key beforehand. IronClad supports client side signing by a tool. It transforms a signing request into a signed response, which is then sent back to the server side.

Once received by the server side (cf. lines 13 to 16), the signed response containing the signatures is verified. Provided that the verification was successful, IronClad stores the signatures in the identity owner's WebID profile in ④. When storing (cf. middle of Figure 2), IronClad applies the method proposed by Sayers and Eshghi in [10]. This method closely links the public key of the profile with the WebID URI. Thus, it assists in detecting attacks that aim at removing profile data and signatures.

Discovering WebID Identity Theft. Following the principle of empowering individuals instead of authorities, we could not solely rely on attaching signatures to profile data[4]. IronClad creates a binding between the public key and the WebID URI. Thus, it ensures that this key cannot be changed without losing personal relationship data such as incoming social connections expressed via `foaf:knows` WebID URIs. Having the public key stored in the WebID URI allows detecting the same key inside the profile. This facilitates not only discovering identity theft done by malicious key manipulation, but also using the public key for signature verification. The length of the public key, e.g., 2048-bit, makes it inconvenient to store it directly inside a WebID URI. Therefore, IronClad uses the SHA-1 hash value of the public key instead.

Algorithm 1. Creating Signatures of User Profile Data

```
   Input: WebID URI $u$, Private Key $key$
   // on server side
1  get WebID profile $p$ from $u$;
2  get RDF graph $g$ from $p$;
3  apply canonicalization on $g$ using [3] and [12];
4  repeat
5      delete minimal self-contained graph $msg$ from $g$;
6      create hash value $h$ of $msg$; // currently uses SHA-1
7      add $h$ to client request $req$;
8  until $g$ is empty;
   // on client side to avoid private key disclosure
9  foreach hash value $h$ in client request $req$ do
10     create signature $sig$ by encrypting $h$ with $key$;
11     add $sig$ to server response $res$;
12 end
   // on server side
13 foreach signature $sig$ in server response $res$ do
14     if signature $sig$ is invalid then stop
15 end
16 add all signatures in server response $res$ to graph $g$;
```

Verifying WebID Profile Data Integrity. To make signed WebID profiles easily verifiable for requestors, we integrated the verification process into the WebID authentication routine. It is triggered when the WebID profile has been loaded. For verifying signed profile data (cf. bottom of Figure 2), IronClad receives the WebID profile via the WebID URI in ⑤. It tries to detect a plausible public key[5] inside the profile. Such public key has to correspond to the hash value stored in the WebID

[4] By gaining access to the system storing the WebID profile, an attacker could tamper identity data and manipulate signatures stored in the profile. Due to this vulnerability to attacks, an external authority would be required to provide proof of correctness.

[5] A public key with a common length, e.g., 2048 bit.

URI. Having found the public key, IronClad computes hash values of WebID profile data as mentioned in the signing process. It then compares the hash values with the hash values retrieved by decrypting the signatures using the public key, as indicated by ⑥. The data integrity of WebID profiles cannot be guaranteed in case any detection or verification step has failed. Handling failed verifications depends on the scenario and authentication target. This is not a part of the IronClad approach and, thus, needs be to addressed separately.

3.2 Implementation and Demonstration of IronClad

For showcasing the approach, we exemplarily implemented IronClad in the Sociddea WebID identity provider and management platform proposed by Wild et al. in [13]. It is important to mention here that IronClad is not limited to a specific platform. That is, it is possible to apply the approach in arbitrary WebID identity providers, management platforms or authentication methods. Thus, IronClad is line with the idea of decentralized social networks.

In Sociddea, IronClad is optionally applied when creating new WebID identities. It is furthermore used for signing and verifying profile data hosted in Sociddea's ecosystem. Figure 3 illustrates the visual representation of a tamper-evident WebID profile in four different scenarios. While the figure shows tampering by changing a personal attribute of the identity owner, data integrity breaches through adding or removing RDF triples are also detected by IronClad. In addition to visually highlighting malicious data manipulations, identity owners and service providers also benefit from tamper-evident WebID profiles during user authentication and requests for profile data.

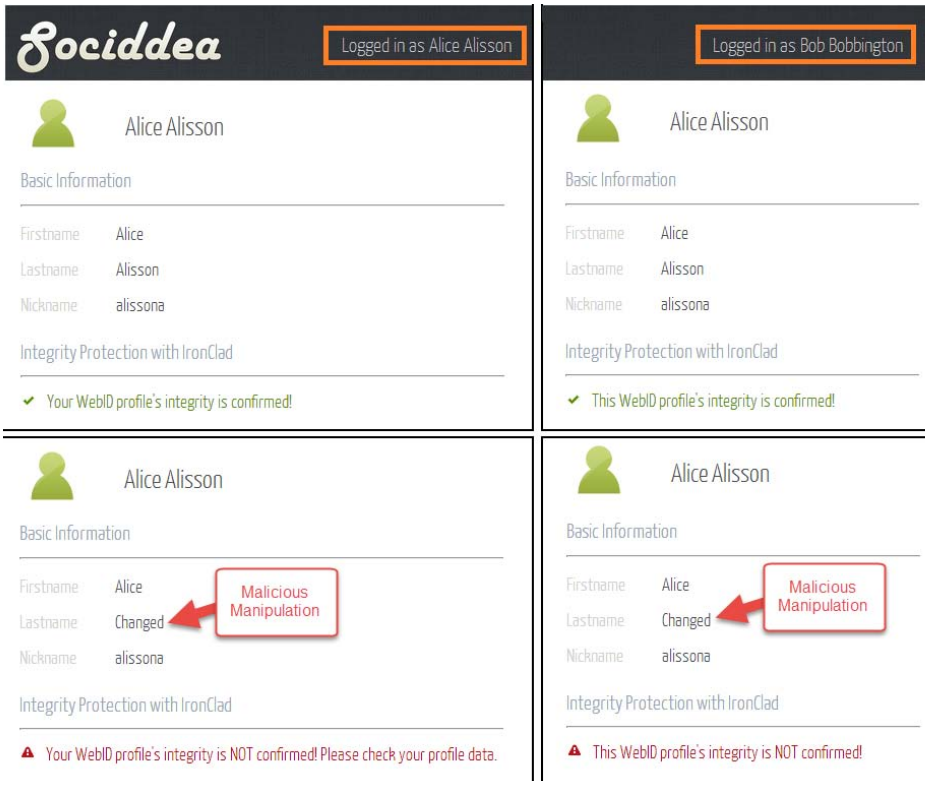
Try It Out. For a live demonstration, screen casts, and further information about IronClad and Sociddea please visit:
`http://vsr.informatik.tu-chemnitz.de/demo/sociddea/`

4 Evaluation of IronClad

While the acceptance and handling of tamper-evident WebID profiles is a planned subject of a larger empirical investigation, this evaluation discusses to which extent IronClad takes the criteria into account we used for analyzing related work. From a theoretical point of view, we think that such study therefore allows for determining how well IronClad achieves the objective defined in Section 1.

WebID profiles depend on RDF for describing an identity owner's personal attributes in a machine-readable way. Appropriate RDF-based vocabularies enable describing and interlinking new contents. This facilitates extending WebID profiles by additional RDF triples. It is consequently well applicable to represent and associate signatures to personal attributes.

As the IronClad approach does not involve encrypting user profile data, it does not impair the accessibility of tamper-evident WebID profiles. Both common and tamper-evident WebID profiles share similar characteristics. While tamper-evident WebID profiles consist of additional RDF-based signatures, existing personal data



Sociddea

Logged in as Alice Alisson

Alice Alisson

Basic Information

Firstname	Alice
Lastname	Alisson
Nickname	alissona

Integrity Protection with IronClad

✓ Your WebID profile's integrity is confirmed!

Logged in as Bob Bobbington

Alice Alisson

Basic Information

Firstname	Alice
Lastname	Alisson
Nickname	alissona

Integrity Protection with IronClad

✓ This WebID profile's integrity is confirmed!

Alice Alisson

Basic Information

Firstname	Alice
Lastname	Changed
Nickname	alissona

Malicious Manipulation

Integrity Protection with IronClad

⚠ Your WebID profile's integrity is NOT confirmed! Please check your profile data.

Alice Alisson

Basic Information

Firstname	Alice
Lastname	Changed
Nickname	alissona

Malicious Manipulation

Integrity Protection with IronClad

⚠ This WebID profile's integrity is NOT confirmed!

Fig. 3. Exemplary implementation of IronClad for tamper-evident user profiles with visualized results for: Identity owner Alice's view on her valid profile data (top/left), "Lastname" changed - tampering detected (bottom/left), requestor Bob's view on Alice's valid profile data (top/right), Bob's view on Alice's tampered profile (bottom/right)

remains untouched. In IronClad, signatures are loosely coupled statements about personal data. Removing all signature RDF triples would reveal the original payload of a WebID profile. The hash value included in the WebID URI, however, indicates that the corresponding WebID profile is tamper-evident. Considering an attacker would remove signatures stored in a tamper-evident WebID profile, there is still the WebID URI indicating that the WebID profile has to be data integrity protected. While there are other ways of using hash values in URIs [9], IronClad just appends them to common WebID URIs for the sake of simplicity and conformity. Such verification for tamper-evidence is part of the proposed extension of the WebID authentication sequence. Even though signatures and personal data contained in tamper-evident WebID profiles are accessible per se, view filters can assist in concealing particular RDF triples, as described by Wild et al. in [14]. For example, excluding all signatures might be beneficial for presenting profile data to users.

Through directly operating on the RDF graph representing a WebID profile, IronClad is compatible to different orders and structures of RDF triples, and diverse types of serialization. This allows for dealing with the high heterogeneity prevailing in this context. Despite all advantages, we also identified some issues restricting the interoperability and compatibility of our approach: Combining the hash value of the public key with the WebID URI creates a universal mean to detect tampering in WebID profiles. This, however, implicates that it is not possible 1) to transform common WebID URIs to tamper-evident ones and 2) to change tamper-evident WebID URIs without invalidating all incoming connections from social contacts expressing their friendship and so on. Changing a WebID URI results in creating a different and therefore new identity. This can be considered as a common shortcoming in WebID. There is a need for an approach to indicate that a new (tamper-evident) WebID URI belongs an already existing one. Creating a tamper-evident WebID profile necessitates signing the profile data worth protecting. As we wanted to avoid disclosing the private key to a third-party, we rely on a client-side signing tool at the moment. This dependency entails new requirements for system, platform and device support. We seek to resolve the dependency by a solution that is user-friendly and interoperable.

5 Conclusion and Future Work

By providing a means for users and machines to detect data integrity breaches in WebID profiles, we contributed to increase the trustworthiness of open, decentralized, and universal identification mechanisms. Taking compatibility and accessibility of user profile data into account, our approach for tamper-evident user profiles is well-applicable for new WebID identities. Focusing on empowering individuals instead of authorities, we expect that such mechanism will gain more momentum and coverage in the future. By adding security features to WebID artifacts, we enabled profile owners and requestors to detect malicious manipulation and identity theft. We integrated IronClad into the WebID authentication routine to verify the profile data integrity as a requirement for any successful identity proof. Not only does IronClad operate independently of data type and order, but also independently of the profile hosting system. We enabled verification without requiring prior knowledge except for an already known WebID identifier.

Having made an important step towards more trustworthiness in the context of WebID, we will constitute future work on that basis. For validating not solely the technical feasibility of our approach, it is necessary to conduct a more comprehensive evaluation involving how well users accept tamper-evident WebID URIs and profiles. Even though we are aware that tamper-evident WebID URIs make the management for users more difficult, e.g., compared to managing email addresses, we claim that most issues can be successfully addressed through utilizing techniques such as URI drag and drop, QR codes or WebID URIs embedded into other objects like WebID certificates. Identifying the origin of tampering attacks, facilitating key replacement and enabling signature creation with several keys are also interesting topics for future research.

References

1. Bamberg, W., et al.: Persona - Protocol Overview (2013), `https://developer.mozilla.org/en-US/docs/Mozilla/Persona/Protocol_Overview` (accessed February 23, 2014)
2. Caronni, G.: Walking the Web of Trust. In: Proeedings of the IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2000), pp. 153–158. IEEE (2000)
3. Carroll, J.J.: Signing RDF Graphs. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 369–384. Springer, Heidelberg (2003)
4. Dhamija, R., Dusseault, L.: The Seven Flaws of Identity Management: Usability and security Challenges. IEEE Security & Privacy 6(2), 24–29 (2008)
5. Feldman, A.J., Blankstein, A., Freedman, M.J., Felten, E.W.: Privacy and Integrity are Possible in the Untrusted Cloud. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 35(4), 73–82 (2012)
6. Feldman, A.J., Blankstein, A., Freedman, M.J., Felten, E.W.: Social Networking with Frientegrity: Privacy and Integrity with an Untrusted Provider. In: Proceedings of the 21st USENIX Conference on Security Symposium, Security, vol. 12, p. 31 (2012)
7. Fitzpatrick, B., Recordon, D., Hardt, D., Hoyt, J.: OpenID Authentication 2.0 - Final (2007), `http://openid.net/specs/openid-authentication-2_0.html` (accessed February 23, 2014)
8. Hackett, M., Hawkey, K.: Security, Privacy and Usability Requirements for Federated Identity. In: Workshop on Web 2.0 Security & Privacy (2012)
9. Sauermann, L., Cyganiak, R., Völkel, M.: Cool URIs for the Semantic Web. Tech. rep., Saarländische Universitäts- und Landesbibliothek (2007)
10. Sayers, C., Eshghi, K.: The case for generating URIs by hashing RDF content (2002)
11. Sporny, M., Inkster, T., Story, H., Harbulot, B., Bachmann-Gmür, R.: WebID 1.0: Web Identification and Discovery (2011), `http://www.w3.org/2005/Incubator/webid/spec/` (accessed February 23, 2014)
12. Tummarello, G., Morbidoni, C., Puliti, P., Piazza, F.: Signing Individual Fragments of an RDF Graph. In: Special Interest Tracks and Posters of the 14th International Conference on WWW, pp. 1020–1021. ACM (2005)
13. Wild, S., Chudnovskyy, O., Heil, S., Gaedke, M.: Customized Views on Profiles in WebID-Based Distributed Social Networks. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 498–501. Springer, Heidelberg (2013)
14. Wild, S., Chudnovskyy, O., Heil, S., Gaedke, M.: Protecting User Profile Data in WebID-Based Social Networks Through Fine-Grained Filtering. In: Sheng, Q.Z., Kjeldskov, J. (eds.) ICWE Workshops 2013. LNCS, vol. 8295, pp. 269–280. Springer, Heidelberg (2013)
15. Yeung, C.M.A., Liccardi, I., Lu, K., Seneviratne, O., Berners-lee, T.: Decentralization: The Future of Online Social Networking. In: W3C Workshop on the Future of Social Networking Position Papers, vol. 2, pp. 2–7 (2009)

X-Themes: Supporting Design-by-Example

Moira C. Norrie, Michael Nebeling, Linda Di Geronimo, and Alfonso Murolo

Department of Computer Science, ETH Zurich
CH-8092 Zurich, Switzerland
{norrie,nebeling,lindad,amurolo}@inf.ethz.ch

Abstract. Design-by-example enables users with little technical knowledge to develop web sites by reusing all or parts of existing sites. In CMS such as WordPress, themes essentially offer example designs for all-or-nothing reuse. We propose an extension to the theme concept that allows web sites to be designed by reusing and combining components of different themes. In contrast to previous research advocating design-by-example, we do not restrict ourselves to static web pages, but also support the reuse of dynamic content including functionality for animations and database access. Our approach is to provide a theme generator that structures the themes that it generates in terms of reusable components which can then be reused in future themes. We present a first prototype tool, called the X-Themes Editor, developed to demonstrate the viability of the approach and investigate requirements and issues. We describe how the X-Themes Editor has been integrated into the WordPress platform as well as discussing the outcomes of these initial investigations.

Keywords: design-by-example, content management system, theme, end-user development, reuse in web design.

1 Introduction

Within the research community, model-driven approaches to web engineering have been promoted as the best way of supporting the systematic development of web sites [1]. Often methods are designed to cater for projects involving large, diverse teams including graphic designers, database architects, programmers and marketing staff.

However, an increasing number of professional as well as personal web sites are being developed by individuals using platforms such as WordPress[1] which allows users to dynamically customise crowdsourced themes. A theme defines a set of templates, stylesheets and media for an entire web site and therefore essentially supports only all-or-nothing reuse.

The idea of this paper is to show how this paradigm of design-by-example could be extended to allow users to design their web sites by reusing and combining features of different themes. For example, they might choose the colour scheme and front page layout of one theme, a slider content component from

[1] http://www.wordpress.org

a second theme, and the drop-down menu style from a third. As discussed in Sect. 2, this approach has been advocated by researchers in the HCI community [2,3], but typically their experiments and studies focus on the static parts of web sites and do not address the technical challenges of extracting and reusing functionality. A key factor in our research was to support the reuse of all aspects of a web site, including animations, client-side processing and database access. Further, to avoid the need for developers to change their work practices, we wanted to investigate how such an approach could be integrated into the WordPress platform.

Our approach, which is presented in Sect. 3, exploits the concept of a theme generator to produce a collection of so-called X-Themes that conform to a well-structured model designed to support reuse. The special X-Themes generated by our tool are then available to other developers who can drag-and-drop components of existing themes into their newly created themes at design time. We outline the main features of a first prototype designed to demonstrate the viability of the approach in Sect. 4 before discussing the main requirements and challenges to be addressed in future research in Sect. 5.

2 Background

Several model-driven approaches to web site development have been proposed, for example WebML [1], Hera [4] and UWE [5]. While these approaches have received acclaim in the research community, they have had limited impact in the web development community at large where many professionals work with a mix of technical knowledge and design skills and build on modern content management systems (CMS) such as WordPress or Drupal[2]. An indication of the scale of the developer community using WordPress is the estimate that 21.9% of the top 10 million web sites are implemented on WordPress[3]. The size and complexity of these web sites varies enormously, but has certainly gone well beyond the original focus on personal blogging sites and now includes online newspapers, e.g. Metro UK[4], e-commerce sites, e.g. Kuborra[5], and information platforms for communities e.g. SAP.info[6].

As highlighted by a recent survey on modern web development practices [6], many of these developers have no formal computer science training and are either self-employed or working in very small organisations. In these settings, developers typically adopt an *interface-driven* approach, starting from a mockup of the interface and first adding client-side functionality before adding server-side functionality by migrating to a CMS platform.

The WordPress platform was developed using a crowdsourcing model that allows their user community to develop and share both *themes* and *plugins.*

[2] `http://www.drupal.org`

[3] `http://www.w3techs.com`, accessed 10 April 2014

[4] `http://metro.co.uk/`

[5] `http://www.kuborra.com/`

[6] `http://en.sap.info/`

A theme is a set of PHP templates, CSS stylesheets and media objects that define the presentation, structure, functionality and types of content of a web site. To develop their own web site, a user simply has to select a theme and then start adding their own content. Themes are usually parametrised so that users can easily customise their sites and they can also extend the types of content and functionality by adding plugins.

Themes can be considered as a form of *design-by-example* [7]. While WordPress themes only support all-or-nothing reuse, researchers have investigated approaches which would allow users to design their web sites by freely selecting and combining parts of example web sites from interactive galleries [2,3]. While these studies have demonstrated the benefits of this general approach, they did not address the technical challenges of being able to extract and combine arbitrary elements of modern web pages that are often dynamic rather than static and make heavy use of JavaScript and jQuery[7]. As a result, their methods were only able to deal with the reuse of appearance and content and not functionality.

A number of approaches for developing web applications from reusable components have been proposed. WebComposition [8] is an object-oriented support system for building web applications through hierarchical compositions of reusable application components. While some mashup editors help users to integrate information from distributed sources, others provide infrastructure for building new applications from reusable components. For example, MashArt [9] enables advanced users to create their own applications through the composition of user interface, application and data components. While our approach shares some of the goals and enables similar extraction and reuse techniques, it offers these at the theme level to base web site designs on multiple examples, which is different from mashups that are direct compositions of existing web sites. More recently, extensions to CMS such as WordPress have been proposed to allow web applications to be developed from a component model that supports composition at the data, application and interface levels [10]. The focus of this work differs in that our approach is *interface-driven* rather than *model-driven* and our component model captures concepts of popular web development platforms such as WordPress as well as the new HTML5 and CSS3 web standards together with jQuery that power the underlying themes.

Summarising, the solution proposed by many researchers has been to try and bring discipline into web engineering by requiring developers to first model different aspects of their web sites and only then generate code. But this usually requires that developers abandon popular platforms and learn new modelling skills, tools and possibly even languages. It also makes it more difficult to support rapid prototyping and allow developers to start by adapting existing web site designs developed either by themselves or other developers. Instead of trying to force developers to change their ways of working, we think it is important to instead find ways of better supporting them. This means that not only should we find ways to support the design-by-example paradigm, but this should be done using existing, popular platforms and technologies.

[7] `http://jquery.com`

3 Approach

While the WordPress platform is very flexible and powerful, the basic model behind it is relatively simple and developers are offered a very loose framework in which to work. As a result, it is possible to build advanced web sites but widely varying approaches are used to achieve similar functionality and presentation.

In the rare case that a personal or professional developer finds an existing theme that fully meets their requirements, the process of developing a web site mainly consists of setting some parameters and adding content through the WordPress dashboard. To some degree, they can even extend the functionality of the theme through this interface by selecting and adding various plugins. However, as soon as a developer is faced with the task of adapting or extending a theme, they have to start working at the level of the HTML, CSS and PHP files and learning about the core WordPress model and system operation. Developers often work on a need-to-know basis, learning only enough to solve the particular task at hand. Further, the documentation and tutorials vary a lot in terms of guidelines and solutions offered to developers. It is clear from reading tutorial-style books, e.g. [11,12] as well as online forums[8] that many developers simply copy and paste bits of CSS, HTML and PHP with the hope that it will achieve the desired effects. However, often these attempts to reuse code fail because they are inconsistent with how other parts of the site have been developed.

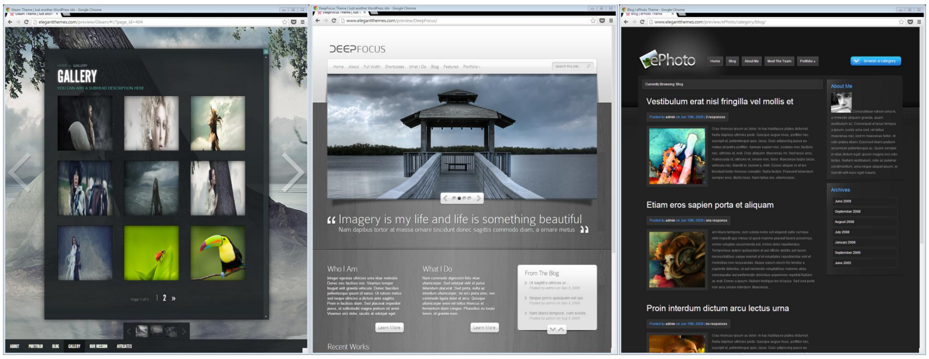

GALLERY
DEEPFOCUS
Imagery is my life and life is something beautiful
Who I Am
What I Do
From The Blog
Recent Works
ePhoto
Vestibulum erat nisl fringilla vel mollis et
Etiam eros sapien porta et aliquam
Proin interdum dictum arcu lectus urna
About Me
Archives

Fig. 1. Examples of portfolio themes from the company Elegant Themes

We want users to be able to create their own themes by selecting and combining parts of different existing themes. For example, Fig. 1 shows three portfolio themes offered by the company Elegant Themes[9]. A user might want to have the general styling of the theme on the left which includes a slider of background images and an animated drop down element in the gallery page, but want to include in the front page the slider component of the theme in the middle and have

[8] For example, `http://www.wpbeginner.com`

[9] `http://www.elegantthemes.com`

a blog post page with the layout shown in the theme on the right. To achieve this, we have developed a graphical theme editor, called the X-Themes Editor, which allows users to select parts of other themes that can be integrated into their own themes using simple drag-and-drop operations.

To support this kind of reuse, it is necessary that themes adhere to a well-defined component model. One way to do this would be to develop a new platform or domain specific language based on a component model but, as stated previously, this is something that we wanted to avoid. Instead, we designed our own metamodel for WordPress themes and then based the X-Themes Editor on this model. This means that any themes generated with our editor are clearly structured in terms of components that can be shared and reused. By offering a theme editor that is integrated into the WordPress platform, developers can already be offered a valuable tool for creating new themes from scratch. Themes generated using the tool are called X-Themes and the collection of X-Themes are made available for reuse in an interactive gallery.

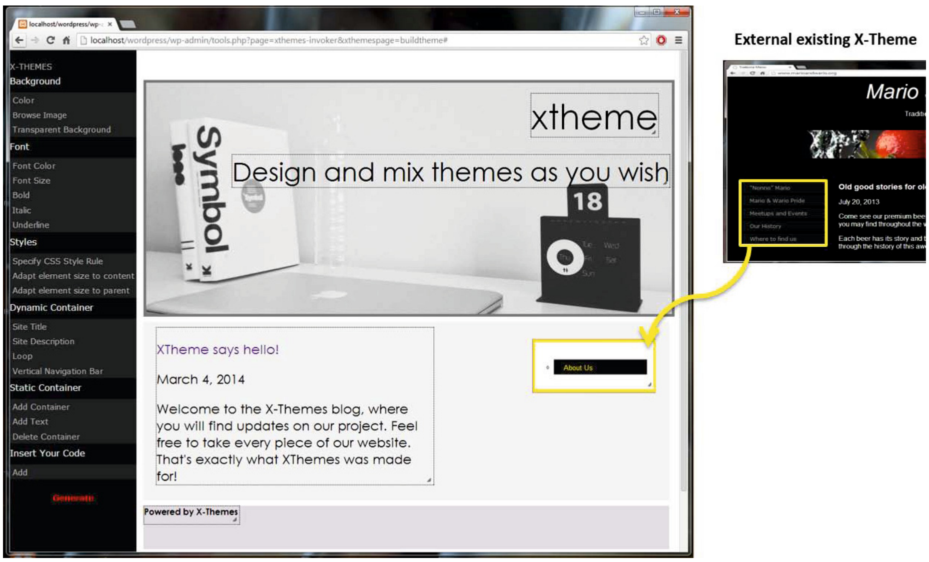
It is important to note that a number of theme generators for WordPress already exist but many of these have serious limitations, especially when it comes to customising functionality. For example, Templatr[10] is a free web-based tool that allows users to customise static elements but they can only select from a fixed set of layouts. Some tools such as Lubith[11] allow users to customise layout via drag and drop, but they usually do not support the customisation of functionality. Another limitation of existing generators is the fact that they are not integrated into WordPress. This means that it is not possible to perform content-related tasks such as displaying the pages or latest posts and comments during the design of the theme and often there can be compatibility problems between WordPress versions. Therefore, offering a theme editor that is integrated into the WordPress platform, and enables not only presentation but also layout and functionality to be customised, is in itself a valuable contribution.

4 X-Themes Editor

Using the possibilities to customise and extend the functionality of the WordPress dashboard, we were able to integrate the X-Themes Editor into the tool options menu offered to developers. When the X-Themes tool is selected, the standard dashboard interface is replaced by the GUI of the X-Themes Editor which, in addition to offering a menu for creating and customising elements of a theme, also provides access to an interactive gallery of previously generated X-Themes. A user can create a theme using a mix of editing operations to define new components and drag-and-drop operations to import selected components from various source themes opened in the X-Themes gallery.

Every theme created in the editor has three main components—Header, Body and Footer— that are visually separated as indicated in Fig. 2. Users can create

[10] `http://templatr.cc`
[11] `http://www.lubith.com`



XTheme says hello!

March 4, 2014

Welcome to the X-Themes blog, where you will find updates on our project. Feel free to take every piece of our website. That's exactly what XThemes was made for!

Fig. 2. Dragging a navigation menu from an existing website into the X-Themes editor

different types of containers within each of these main components and each container may itself contain other containers.

As soon as a component from an existing theme is dropped into the new theme, it will be active and any dynamic behaviour experienced. For example, if the user drag-and-drops a dropdown menu into the navigation component as illustrated in Fig. 2, they will immediately be able to try out this functionality in the theme under construction.

There are two main types of container—static and dynamic. A static container has no elements that can change dynamically. A dynamic container is one in which at least one JavaScript or PHP function appears. This could be a function to access a theme parameter or content of the database such as the site's title or application data. A container with any kind of dynamic content will be executed and show the corresponding result already at design time, even when dragged-and-dropped from a different theme.

After a drag-and-drop operation, the user can choose whether or not they want to keep the style of the source theme or have the style of the destination theme applied. For example, a user may wish to keep the information displayed in an imported database query (referred to as a Loop in WordPress) but apply a different design in the new theme. If the user decides to keep the style of the source theme, they can adjust both the format and style later using the general editing functionality of the X-Themes Editor.

When the editing of a theme is complete, the X-Themes Editor generates the set of templates for that theme together with the files defining the components of the X-Theme and associated metadata based on the X-Themes metamodel shown in Fig. 3.

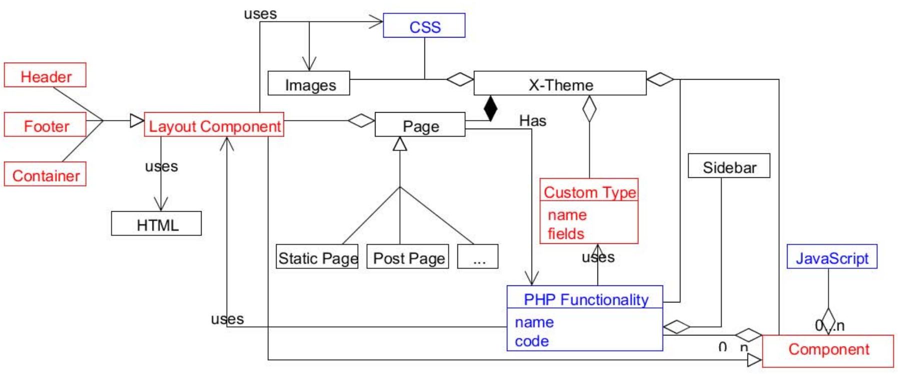


Fig. 3. X-Themes metamodel

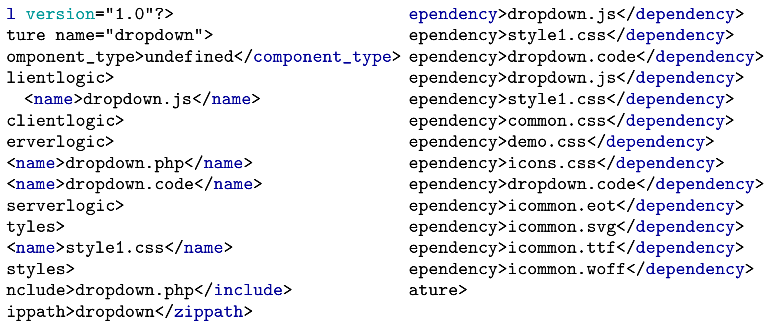
Concepts defining basic structural elements of a web page such as header, footer and sidebar are fundamental to the WordPress model as are posts as the primary content type. To extend the model to support other types of content, custom post types can be introduced. Our metamodel introduces the concept of a component shown at the bottom right of Fig. 3. A component can include layout and style rules as well as dynamic behaviour specified as PHP logic or JavaScript. If a component is to be exported and reused, then clearly all code, including CSS rules as well as PHP logic and JavaScript, must also be exported along with the component.

To support the extraction and reuse of components, each component is described in an XML file of the form shown in Fig. 4. All the required PHP, CSS and JavaScript files are stored in a separate folder in the WordPress theme directory, and the DOM of the theme is annotated using HTML5 dataset properties to indicate reusable components based on the metamodel.

The `component_type` can be custom header, sidebar or undefined in the case of a generic component. For header and sidebar components, a `functions` element is included to specify dependencies so it is known which files have to be included in the *functions.php* file of a WordPress theme for that component to be used.

A `clientlogic` element specifies JavaScript code that needs to be exported with the component, while `serverlogic` does the same for PHP logic. Any required style files are specified in the *styles* element. The `include` element specifies the file to be included in a page of the theme to execute and show the component. Various forms of dependencies to other code files are specified in the dependency elements.

A `layout_component` is a special type of component representing the layout structure of the theme and can be a header, a footer or a container. Any of these three types of component can include containers, allowing a fully nested container structure. We distinguish layout components that contain only static content from those that include JavaScript or PHP code.



```
l version="1.0"?>
ture name="dropdown">
omponent_type>undefined</component_type>
lientlogic>
  <name>dropdown.js</name>
clientlogic>
erverlogic>
<name>dropdown.php</name>
<name>dropdown.code</name>
serverlogic>
tyles>
<name>style1.css</name>
styles>
nclude>dropdown.php</include>
ippath>dropdown</zippath>
```

```
ependency>dropdown.js</dependency>
ependency>style1.css</dependency>
ependency>dropdown.code</dependency>
ependency>dropdown.js</dependency>
ependency>style1.css</dependency>
ependency>common.css</dependency>
ependency>demo.css</dependency>
ependency>icons.css</dependency>
ependency>dropdown.code</dependency>
ependency>icommon.eot</dependency>
ependency>icommon.svg</dependency>
ependency>icommon.ttf</dependency>
ependency>icommon.woff</dependency>
ature>
```

Fig. 4. Component XML Sample

Each component is allocated a separate directory where its code, resources and XML component files are stored. This includes all PHP required to reuse the component as well as generated CSS files. In the final step of theme generation, an XML representation of the theme defining the component structure is created.

When a user opens a theme in the X-Themes gallery, reusable components are highlighted as the cursor moves over them. If the user drags a component, such as a navigation menu made in PHP/JavaScript and CSS, the X-Themes Editor accesses the metamodel information of that component and acquires references to linked resources which are then cloned for use in the editor. The component is then executed in the context of the X-Themes Editor running on the user's own WordPress installation. The ability for users to view their own content at design time is only one of the advantages of implementing the X-Themes Editor as a WordPress plugin. Another is the fact that it provides an easy means of deploying the tool to existing developer communities and hence raising awareness and chances of acceptance.

5 Discussion

While the X-Themes Editor represents an important first step in showing how design-by-example could be fully supported in CMS platforms such as WordPress, there are a number of issues that need to be addressed in future research.

X-Themes Metamodel. The metamodel that we have worked with so far is based on the WordPress core model and therefore WordPress specific. Further, it focusses more on system and implementation features than general asbtract concepts. One of the next steps will be to generate a conceptual metamodel that could be applied to more than one CMS. Specifically, we are currently investigating how our approach could be applied in Drupal and the metamodel generalised.

Beyond the Front Page. Although the themes that we currently generate are not single page, like many WordPress themes, they very much focus on the front page which is often seen as defining the main features of a web site in terms of the header, footer, navigation and layout as well as presentation. We need to extend the X-Themes Editor to cater for web sites with more complex structures with variable page layout and content.

Transforming Themes to X-Themes. One of the main ideas behind our approach was to provide users with a graphical theme editor that would in itself be a valuable tool and hence something that developers would want to use independent of the goals of this project. In this way, we could address the cold start problem of generating a collection of X-Themes available for reuse. At the same time, we do think it important that we offer support for transforming existing WordPress themes into X-Themes. This is a topic for future research where we will investigate the extent to which this could be automated.

Interactive Galleries. The current way of searching for WordPress themes relies on descriptions and keywords provided by developers. We want to investigate alternative and complementary ways of supporting search within interactive galleries. This would include search-by-example as well as ways of automatically classifying themes based on a variety of factors.

Data-Intensive Web Sites. Data-intensive web sites require the integration of custom post types to manage application data. In previous work within our research group, a tool was developed that generates a WordPress plugin with custom post types based on an entity-relationship data model defined by a developer [13]. We have now started to investigate an alternative approach that would extend the design-by-example approach to automatically derive data schemas and custom post type definitions based on example data content.

6 Conclusion

We have shown how the support for design-by-example offered by modern CMS platforms such as WordPress could go well beyond the current all-or-nothing approach of sharing entire themes. Users should be able to selectively reuse and combine parts of existing themes, including dynamic components that define functionality. Our work therefore goes beyond previous research on design-by-example paradigms in web development [2,3] which were limited to static views on web sites.

We note however that, while the X-Themes Editor we have developed is sufficient to demonstrate the potential of the approach, as outlined in Sect. 5, there are a number of open issues that would need to be addressed in future research in order for the method to be deployed in practice. Further, the tool would need to advance beyond the prototype stage to support a full palette of editing capabilities expected of a state-of-the-art web design tool.

Acknowledgements. We acknowledge the support of the Swiss National Science Foundation who financially supported this research under project FZFSP0_147257.

References

1. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann Publishers Inc. (2002)
2. Hartmann, B., Wu, L., Collins, K., Klemmer, S.R.: Programming by a Sample: Rapidly Creating Web Applications with d.mix. In: Proc. 20th ACM User Interface Software and Technology Symposium, UIST (2007)
3. Lee, B., Srivastava, S., Kumar, R., Brafman, R., Klemmer, S.: Designing with Interactive Example Galleries. In: Proc. Conf. on Human Factors in Computings Systems, CHI (2010)
4. Houben, G., Barna, P., Frasincar, F., Vdovjak, R.: Hera: Development of Semantic Web Information Systems. In: Cueva Lovelle, J.M., Rodríguez, B.M.G., Gayo, J.E.L., Ruiz, M.d.P.P., Aguilar, L.J. (eds.) ICWE 2003. LNCS, vol. 2722, pp. 529–538. Springer, Heidelberg (2003)
5. Hennicker, R., Koch, N.: A UML-based methodology for hypermedia design. In: Evans, A., Caskurlu, B., Selic, B. (eds.) UML 2000. LNCS, vol. 1939, pp. 410–424. Springer, Heidelberg (2000)
6. Norrie, M.C., Geronimo, L.D., Murolo, A., Nebeling, M.: The Forgotten Many? A Survey of Modern Web Development Practices. In: Casteleyn, S., Rossi, G., Winckler, M. (eds.) ICWE 2014. LNCS, vol. 8541, pp. 285–302. Springer, Heidelberg (2014)
7. Herring, S., Chang, C., Krantzler, J., Bailey, B.: Getting Inspired! Understanding How and Why Examples are Used in Creative Design Practice. In: Proc. Conf. on Human Factors in Computings Systems, CHI (2009)
8. Gellersen, H.W., Wicke, R., Gaedke, M.: WebComposition: An Object-Oriented Support System for the Web Engineering Lifecycle. Computer Networks 29(8-13) (1997)
9. Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F., Matera, M.: A Framework for Rapid Integration of Presentation Components. In: Proc. 16th Intl. World Wide Web Conference, WWW (2007)
10. Leone, S., de Spindler, A., Norrie, M.C., McLeod, D.: Integrating Component-Based Web Engineering into Content Management Systems. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 37–51. Springer, Heidelberg (2013)
11. Blakeley-Silver, T.: WordPress Theme Design. Packt Publishing (2008)
12. Casabona, J.: Building WordPress Themes from Scratch (2012)
13. Leone, S., de Spindler, A., Norrie, M.C.: A Meta-plugin for Bespoke Data Management in WordPress. In: Wang, X.S., Cruz, I., Delis, A., Huang, G. (eds.) WISE 2012. LNCS, vol. 7651, pp. 580–593. Springer, Heidelberg (2012)

A Tool for Detecting Bad Usability Smells in an Automatic Way

Julián Grigera[1], Alejandra Garrido[1,2], and José Matías Rivero[1,2]

[1] LIFIA, Facultad de Informática, Universidad Nacional de La Plata, Argentina
[2] CONICET, Argentina
{Julian.Grigera,Garrido,MRivero}@lifia.info.unlp.edu.ar

Abstract. The refactoring technique helps developers to improve not only source code quality, but also other aspects like usability. The problems refactoring helps to solve in the specific field of web usability are considered to be issues that make common tasks complicated for end users. Finding such problems, known in the jargon as *bad smells*, is often challenging for developers, especially for those who do not have experience in usability. In an attempt to leverage this task, we introduce a tool that automatically finds bad usability smells in web applications. Since bad smells are catalogued in the literature together with their suggested refactorings, it is easier for developers to find appropriate solutions.

1 Introduction

The refactoring technique [1] has been recently brought to the usability field of web applications, allowing developers to apply usability improvements without altering the application's functionality [2]. The problems developers can solve by refactoring, called *bad usability smells* are often hard to find, so there are ways to assist this task.

Running usability tests is the most common way to find usability problems [3], but it requires supervision by usability experts, among other resources. These tests can be automated to some extent, presenting an attractive alternative to lower the costs. Some approaches in the literature automate the gathering of data from users [4, 5] but not the analysis, which still depends on usability experts. Other approaches automate part of the analysis by comparing users behavior to optimal behavior paths [6, 7], but this requires prior preparation and subjects to conduct the experiments. There are also commercial tools like CrazyEgg[1] or ClickTale[2] that offer statistical data to their customers by analyzing interaction data from real users instead of tests subjects. However, even if these tools can represent a cheaper option, the results they obtain also require analysis from usability experts.

The tool we present in this work also automates the gathering of interaction data from real users, but in addition, it preprocesses the events on the client side to report concrete usability problems, easier to interpret than mere statistics. Moreover, the tool presents the usability issues as *bad usability smells*, which are problems catalogued in the literature along with the *refactorings* that solve them. Using these catalogues, developers can find a

[1] http://www.crazyegg.com
[2] http://www.clicktale.com

concrete way to correct the detected bad smells. The *Bad Smells Finder* (as we called our tool) was developed as the first stage of a process for automatically improving usability on web applications. We explain this process in the next section.

2 The Process in a Nutshell

Our process for improving web usability is based on the refactoring technique. When applying refactoring in the context of web usability, developers first must detect bad usability smells, and then they must find refactorings to solve them, keeping the basic functionality intact. The tool helps them find bad usability smells.

The automated process for finding bad smells consists in three steps, depicted in Fig. 1. The **Threats Logger** is a client-sided script that gathers interaction events from real users. Instead of logging raw, atomic events, it processes them to generate *usability threats*, a concept we devised to represent higher-level interaction events.

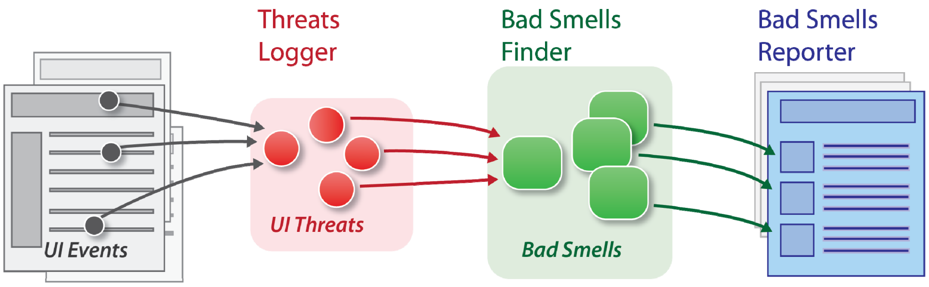


Fig. 1. Schematics of the process

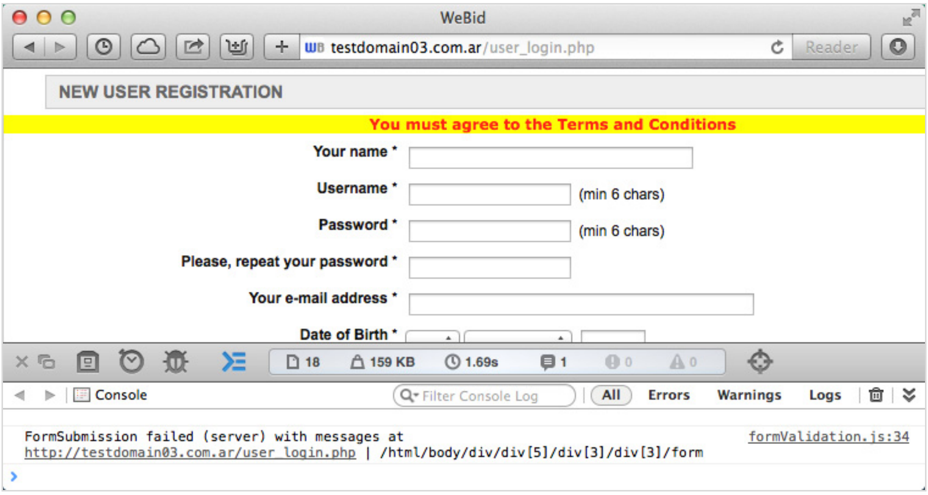
The server-sided **Bad Smells Finder** receives usability threats and stores them for analysis. When a user requests a report, the Bad Smells Finder processes the threats and displays the resulting bad smells through the **Bad Smells Reporter** frontend.

3 The Tool in Action

We will show how the tool works with an example. Consider for instance a web application where users need to register before they can operate. Whenever a user fills the registration form, the threats logger captures the form submission event and evaluates what happens next:

- If no navigation follows, the threats logger considers the submission was blocked by **client-side validation**.
- If a navigation is detected, and the form is still on the destination page, the logger considers there was **server-side validation**.
- If a navigation is detected, but the form is absent in the destination page, the logger considers a **successful submission**.

The tool uses a simple algorithm to identify search forms, where validation rules do not generally apply. Combining all this information on the client-side, it creates a *Form Submission* threat and sends it to the Bad Smells Finder. The script can be set to *verbose* mode to show the threats it finds in the browser's console, as seen in Fig. 2.



```
FormSubmission failed (server) with messages at                    formValidation.js:34
http://testdomain03.com.ar/user_login.php | /html/body/div/div[5]/div[3]/div[3]/form
```

Fig. 2. Threats Logger indicating the detection of a *Failed Submission* threat

The server-side Bad Smells Finder processes all Form Submission threats to potentially find a *No Client Validation* bad smell, which indicates a problematic form that usually fails to submit without offering any client-side validation whatsoever. To do this, it compares the amount of successful submissions with the ones that failed with server validation, according specific criteria for the proportion threshold (e.g. 30% of failed validations indicate a bad smell).

The site owners may then ask for a report by accessing the tool's Reporter, where bad smells are listed with data like the URL where it happened, an XPath of the affected element, and specific extra information depending on each bad smell.

The Bad Smells Finder can detect 12 different kinds of bad usability smells, and the logics for detecting each one are diverse. Other featured bad smells are:

- **No Processing Page:** By calculating the average time of a request and watching DOM mutations, the Bad Smells Finder is able to detect that a process usually takes a long time, but users are never informed about that process taking place on the background (i.e. "loading…" widget).
- **Unnecessary Bulk Action:** Users perform actions on a list of items by first marking checkboxes, and then selecting the action – e.g. deleting emails on a webmail application. If the Bad Smells Finder detects that most of the time users apply actions one item at a time rather than many, then the Unnecessary Bulk Action is detected, implying that the UI with checkboxes and actions should be complemented with other mechanism that require less interactions.
- **Free Input For Limited Values:** A free text input is presented to the user, but the set of possible values that can be entered belong to a limited set, like countries or occupations. Two problems ensue: error-proneness, and unnecessarily time wasted in typing the whole text. The Bad Smells Finder captures all the inputs and calculates the proportion of repeated (and similar) values, in order to determine the bad smell's presence.

The rest of the bad smells are related to navigation issues (like long paths for frequently accessed pages), and misleading/misused widgets. We are currently extending the tool to detect more bad usability smells.

4 Tool Implementation and Usage

The tool has two main modules: the Threats Logger, implemented as a client-side script, and the Bad Smells Finder, a server-sided component that analyzes threats and reports bad smells.

The client-side Threats Logger (coded in JavaScript using the JQuery[3] library) captures interaction events and then processes them on the client to create (and filter) usability threats. The server-sided analyzer parses the incoming asynchronous POST requests from the client script to generate usability threats. When a report is asked, the analyzer filters all the threats to find potential bad usability smells, and then renders the bad smells report in the web frontend.

To install the tool, the site owner must include the Threats Logger script in the application's header. After completing this step, the Bad Smells Finder starts logging and reporting bad usability smells right away.

References

1. Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D.: Refactoring: Improving the Design of Existing Code. Object Technology Series. Addison Wesley (1999)
2. Garrido, A., Rossi, G., Distante, D.: Refactoring for Usability in Web Applications. IEEE Softw. 28, 60–67 (2011)
3. Rubin, J., Chisnell, D.: Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests. Wiley (2008)
4. Atterer, R., Wnuk, M., Schmidt, A.: Knowing the user's every move. In: Proceedings of the 15th International Conference on World Wide Web, WWW 2006, p. 203. ACM Press, New York (2006)
5. Saadawi, G.M., Legowski, E., Medvedeva, O., Chavan, G., Crowley, R.S.: A Method for Automated Detection of Usability Problems from Client User Interface Events AMIA 2005 Symposium Proceedings, pp. 654–658 (2005)
6. Fujioka, R., Tanimoto, R., Kawai, Y., Okada, H.: Tool for detecting webpage usability problems from mouse click coordinate logs. In: Jacko, J.A. (ed.) HCI 2007. LNCS, vol. 4550, pp. 438–445. Springer, Heidelberg (2007)
7. Okada, H., Fujioka, R.: Automated Methods for Webpage Usability & Accessibility Evaluations. In: Adv. Hum. Comput. Interact, ch. 21, pp. 351–364. In-Tech Publ. (2008)

[3] `http://jquery.com`

An Extensible, Model-Driven and End-User Centric Approach for API Building

José Matías Rivero[1,2], Sebastian Heil[3], Julián Grigera[1], Esteban Robles Luna[1], and Martin Gaedke[3]

[1] LIFIA, Facultad de Informática, UNLP, La Plata, Argentina
{mrivero,julian.grigera,esteban.robles}@lifia.info.unlp.edu.ar
[2] Also at Conicet
[3] Department of Computer Science, Chemnitz University of Technology, Germany
{sebastian.heil,martin.gaedke}@informatik.tu-chemnitz.de

Abstract. The implementation of APIs in new applications is becoming a mandatory requirement due to the increasing use of cloud-based solutions, the necessity of integration with ubiquitous applications (like Facebook or Twitter) and the need to facilitate multi-platform support from scratch in the development. However, there is still no theoretically sound process for defining APIs (starting from end-user requirements) or their productive development and evolution, which represents a complex task. Moreover, high-level solutions intended to boost productivity of API development (usually based on Model-Driven Development methodologies) are often difficult to adapt to specific use cases and requirements. In this paper we propose a methodology that allows capturing requirements related to APIs using end-user-friendly artifacts. These artifacts allow quickly generating a first running version of the API with a specific architecture, which facilitates introducing refinements in it through direct coding, as is commonly accomplished in code-based Agile processes.

Keywords: API, Model-Driven Development, Agile Development, Prototyping.

1 Introduction

Over the last years, users and businesses have witnessed a trend to *move* applications and services to the cloud. Several aspects motivate this trend, most importantly cost and deployment time. In this context, developers must interact with applications and services they do not directly control, so they need APIs to facilitate this interaction. Since APIs generally centralize operations and business-logic among applications in several platforms, they are inherently complex; however, most development processes do not take this into account, particularly Agile methodologies, which do not provide clear and structure method to cope with the complexity of API design [1]. To tackle such complex requirements, we have introduced a Model-Driven Development (MDD) [2] solution called *MockAPI* [1], which is limited to providing a prototypical version of the API. In this paper we propose *ELECTRA (*standing for ***E****xtensible mod-e****L****driven* ***E****nduser* ***C****en****TR****ic* ***A****PI)*, an hybrid Agile, MDD and coding approach that

(1) uses an end-user friendly language to define API-related requirements as in MockAPI (annotated mockups), (2) allows to quickly define and generate a running API for testing integration with other software artifacts, and (3) proposes an API runtime that can be extended (hence the term *Extensible*) with custom code without breaking the model's abstraction. We chose mockups as our main requirement artifact because of their positive results in agile approaches [3], and their valuable requirements communication capabilities [4]. Using mockup and annotations as an end-user friendly language we intend to capture the complex API requirements and, at the same, time, provide a framework for quick API generation and refining.

2 The ELECTRA Approach

The ELECTRA process, depicted in Figure 1, is an adaptation of Scrum [5], the most widely used agile process in industry [6]. As in Scrum, every iteration in the ELECTRA approach starts by selecting the User Stories to be tackled (*Define Sprint Backlog* step). Then, ELECTRA mandatorily requires building mockups with essential end-user participation to concretize each of these stories (*Mockup Construction* step). After all User Stories are associated to mockups, developers use an enhanced version of the MockAPI tool [1] to tag the mockups with API-related annotations (*Mockup Tagging* step in Figure 1). These annotations are based on a simple grammar that makes them easy to understand by end-users. From the annotated mockups, an API can be derived through a code generation process (*API Generation* step). At this point, developers get a running usable API for integration testing with other software artifacts (*API testing* step). The generated API is deployed to ELECTRA's runtime environment which allows developers to refine it through direct coding (*API Refining*). The result of this process is the Final API for the current iteration (*API Increment* step).

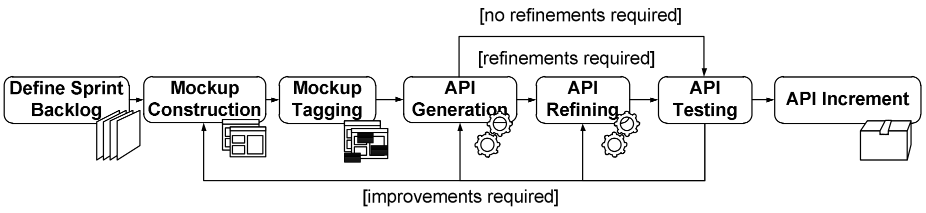
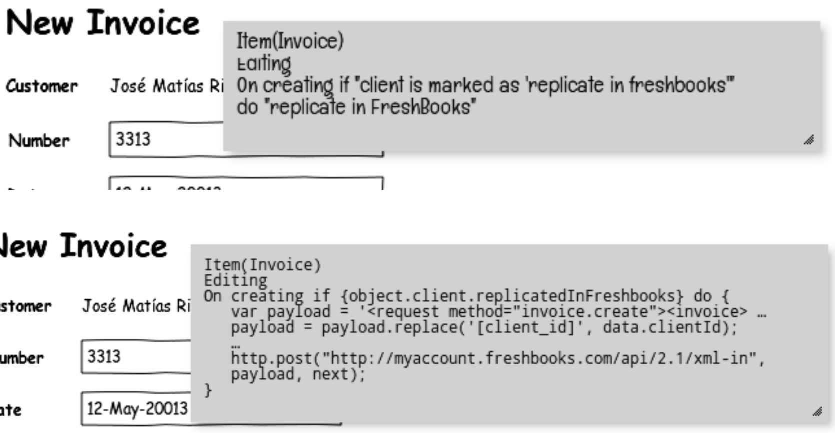


Fig. 1. The ELECTRA workflow

Any type of user interface mockup can be used with ELECTRA tooling. After mockups have been imported, different kinds of annotations can be defined by the engineers in presence and with collaboration of end-users. The three most important annotations types are *Data annotations*, which allow defining object types or business entities well-known by stakeholders, *Constraint annotations*, which enable the definition of business rules and *Action annotations*, which describe the execution of heterogeneous or complex tasks within the API. Besides its technical specifications (understood by engineers), some annotations provide an end-user friendly structured text mode to describe API requirements in natural language. While end-users require engineers help to write annotations, this text mode eases their understanding. ELECTRA tooling currently uses JavaScript as its default scripting language, but any scripting language implementing the Java Scripting API can be used instead. In Figure 2 a *Data* and *Action* annotations are shown applied over an invoice management application to specify the existence of an Invoice business object and how it must be integrated with an external API.



```
Item(Invoice)
Editing
On creating if "client is marked as 'replicate in freshbooks"
do "replicate in FreshBooks"
```

```
Item(Invoice)
Editing
On creating if {object.client.replicatedInFreshbooks} do {
   var payload = '<request method="invoice.create"><invoice> …
   payload = payload.replace('[client_id]', data.clientId);
   …
   http.post("http://myaccount.freshbooks.com/api/2.1/xml-in",
   payload, next);
}
```

Fig. 2. Annotations in its different modes: end-user friendly (upper) and developer (lower)

3 Architecture and Code Generation

The core of the ELECTRA tool implementation (the so-called ELECTRA API runtime) defines and implements a RESTful API as a set of Endpoints, which are composed by an HTTP method, a URL regular expression and a script. When a new HTTP request is received by the runtime, it seeks for a matching Endpoint (with the same HTTP method and a matching URL regular expression). If it finds one, it executes its internal script. Endpoint scripts can access and invoke other *Scripts*, use stored *Resources* (for instance, media content as images or video) or invoke operations on *Services,* which are declared and customized by developers.

ELECTRA's code generation process consists in the generation of a set of Endpoints and Scripts, and the configuration of a default DB Service for storage purposes analyzing the annotated mockups. After triggering a code generation, developers can tune the API as much as needed just editing the required Scripts and Endpoints or changing the DB Service used. Also, they can define new Resources, Services or Endpoints manually. Scripts are generated respecting the Pipes and Filters pattern (where every Filter is formed by a Script implementing a different concern – i.e., code for a different annotation type –, thus facilitating and isolating changes in the API.

When triggering a code regeneration, edited Scripts (Filters) are not altered, thus preserving changes incorporated through direct coding.

4 Related Work

Benefits of using mockups as a primary requirement artifact in the development process have been reported through statistical studies [7]. Also, its benefits in the context of Agile and MDD processes (even API generation) have been commented [8]. In addition, in our previous work [8] we demonstrated that mockup and annotations provide a modeling framework that results more efficient than manual modeling, even considering conceptual models – which are very similar to the *Data annotations* presented in this work. However, none of these works propose a Model-Driven approach that can be extended through direct coding even in the models, as ELECTRA provides. Several MDD approaches specifically tackle RESTful APIs [9,10] (as ELECTRA), but they not provide a clear and pattern-based codebase allowing the quick introduction of detailed implementation as ELECTRA does.

References

[1] Rivero, J.M., Heil, S., Grigera, J., Gaedke, M., Rossi, G.: MockAPI: An Agile Approach Supporting API-first Web Application Development. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 7–21. Springer, Heidelberg (2013)

[2] Kelly, S., Tolvanen, J.-P.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society (2008)

[3] Ferreira, J., Noble, J., Biddle, R.: Agile Development Iterations and UI Design. In: Agil. 2007 Conf., pp. 50–58. IEEE Computer Society, Washington, DC (2007)

[4] Mukasa, K.S., Kaindl, H.: An Integration of Requirements and User Interface Specifications. In: 6th IEEE Int. Requir. Eng. Conf., pp. 327–328. IEEE Computer Society, Barcelona (2008)

[5] Sutherland, J., Schwaber, K.: The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process

[6] VersionOne Inc., State of Agile Survey (2011)

[7] Ricca, F., Scanniello, G., Torchiano, M., Reggio, G., Astesiano, E.: On the effectiveness of screen mockups in requirements engineering. In: 2010 ACM-IEEE Int. Symp. Empir. Softw. Eng. Meas. ACM Press, New York (2010)

[8] Rivero, J.M., Grigera, J., Rossi, G., Luna, E.R., Montero, F., Gaedke, M.: Mockup-Driven Development: Providing agile support for Model-Driven Web Engineering. Inf. Softw. Technol., 1–18 (2014)

[9] Pérez, S., Durao, F., Meliá, S., Dolog, P., Díaz, O.: RESTful, Resource-Oriented Architectures: A Model-Driven Approach, in: Web Inf. In: Chen, L., Triantafillou, P., Suel, T. (eds.) WISE 2010. LNCS, vol. 6488, pp. 282–294. Springer, Heidelberg (2010)

[10] Valverde, F., Pastor, O.: Dealing with REST Services in Model-driven Web Engineering Methods. In: V Jornadas Científico-Técnicas En Serv. Web y SOA, JSWEB (2009)

Building Bridges between Diverse Identity Concepts Using WebID

Michel Rienäcker, Stefan Wild, and Martin Gaedke

Technische Universität Chemnitz, Germany
{firstname.lastname}@informatik.tu-chemnitz.de

Abstract. Single sign-on systems enable users to log into different Web services with the same credentials. Major identity providers such as Google or Facebook rely on identity concepts like OpenID or OAuth for this purpose. WebID by the W3C offers similar features, but additionally allows for storing identity data in an expressive, extensible and machine-readable way using Linked Data. Due to differences in manageable user attributes and the authentication protocols as such, the identity concepts are incompatible to each other. With more than one identity concept in use, users need to remember or keep further credentials. In this paper we therefore propose the B3IDS approach. It aims at improving the user experience and the adoption of WebID by building bridges between diverse identity concepts with WebID. We exemplarily implement B3IDS in an existing WebID identity provider and management system.

Keywords: Identity, Linked Data, Social Networks, Security, WebID.

1 Introduction

For providing a personalized experience, today's Web applications rely on user authentication. Major single sign-on systems have gradually replaced proprietary authentication solutions by consolidating the users' digital identities. So, they also addressed the password-fatigue issue [1]. Global enterprises, e.g., Google or Facebook, offer widely used authentication systems like OpenID or Facebook Connect. While this is convenient for the users, it carries the risk of analyzing user-generated content, including shared information, messages and votes. This enables tracking the user's behavior and creating tailored advertisements. To prevent this risk, the W3C created WebID, which does not only enable authentication, but also empowers people to keep control about their identity data [3].

Despite the advantages, WebID is still in development and not yet broadly accepted by common Web users. Service providers are also reluctant to integrate it as an authentication option. Reasons might be the missing involvement of username/password pairs and technological problems through requesting a certificate. An insufficient adoption by service providers reduces the chance of convincing more users from the benefits of WebID and, thus, prevents its wider use.

Enabling users to access their Web applications and services as usual and make use of WebID requires the availability of identity bridges. Identity bridging is predicted by Gartner as a major trend in the next year [4]. Existing approaches including Mozilla BigTent (`https://wiki.mozilla.org/Identity/BrowserID/BigTent`), CA CloudMinder Gateway [2] and ForgeRock Bridge Service [6] enable bridging between particular identity concepts such as OpenID and Mozilla Persona, but they are characterized by limitations including only providing one-way identity bridging or only enabling access to selected applications via SAML or OAuth.

In order to address the problem, we propose the B3IDS approach. It uses WebID to mediate between different identity concepts. In WebID, all identity data is based on RDF. So, it is expressive, extensible, and machine-readable. This enables representing all data required in other identity concepts in so-called WebID profiles. Users maintain exclusive control about their identity data.

To demonstrate the approach, we show how B3IDS mediates between WebID and OpenID as an example. Relying on WebID, B3IDS strives to bridge the gap between diverse identity concepts by considering 1) exchange of identity data, 2) authentication as usual, and 3) WebID creation from existing identities like an OpenID. So, B3IDS aim at giving rise to the adoption of WebID by the users.

2 Building Secure Bridges to WebID

For bridging between diverse identity concepts, B3IDS mimics two generic components: an identity provider and a relying party. The B3IDS identity provider allows users for authenticating to relying parties of other identity concepts using their WebID certificates as specified in the WebID authentication sequence. Relevant data available in a WebID profile is therefore transparently mapped to data required in the other identity concept. The B3IDS relying party enables users to create a new WebID identity based on their identity data from another identity concept like OpenID. Users can then authenticate to services with such WebID identities. As an example, Alice wants to access a WebID service, but only has a Google account. To get access, she uses B3IDS to issue an authentication request to Google. Alice authenticates herself and, thus, authorizes B3IDS to retrieve identity data from Google. The approach then creates a new WebID profile and a WebID certificate which Alice can use for logging into the WebID service.

B3IDS also incorporates logging Alice into a relying party of another identity concept like OpenID, as shown in Fig. 1. Therefore, B3IDS mimics an OpenID provider retrieving user data from Alice's WebID profile. To accomplish this, Alice passes an identifier to the OpenID relying party (cf. ① in Fig. 1) . The identifier contains the URI of the location of the identity bridge and the WebID URI linking to her WebID profile in the query part, e.g., `https://b3ids.example.org/?webid=https://webid.example.org/Alice`. The OpenID relying party redirects Alice's user agent to B3IDS using the given identifier. It then requests her identity data (cf. ②). Having received Alice's

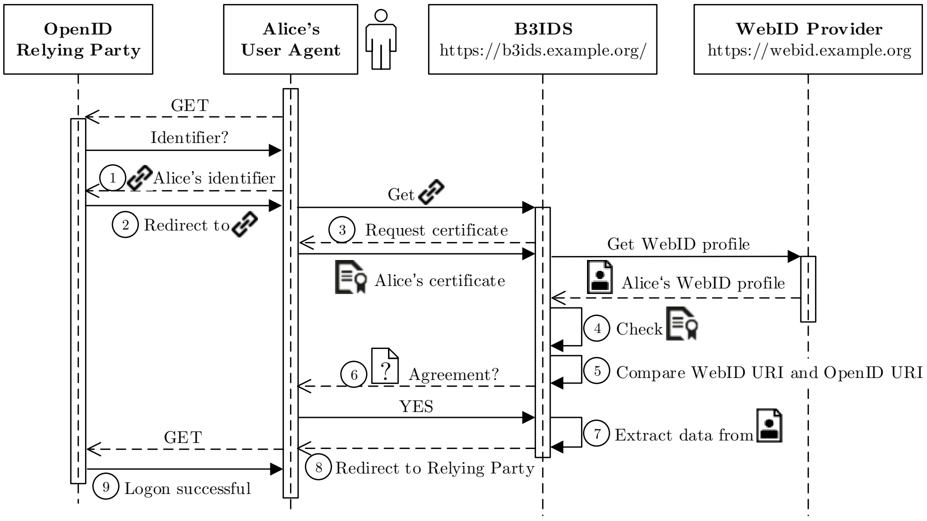
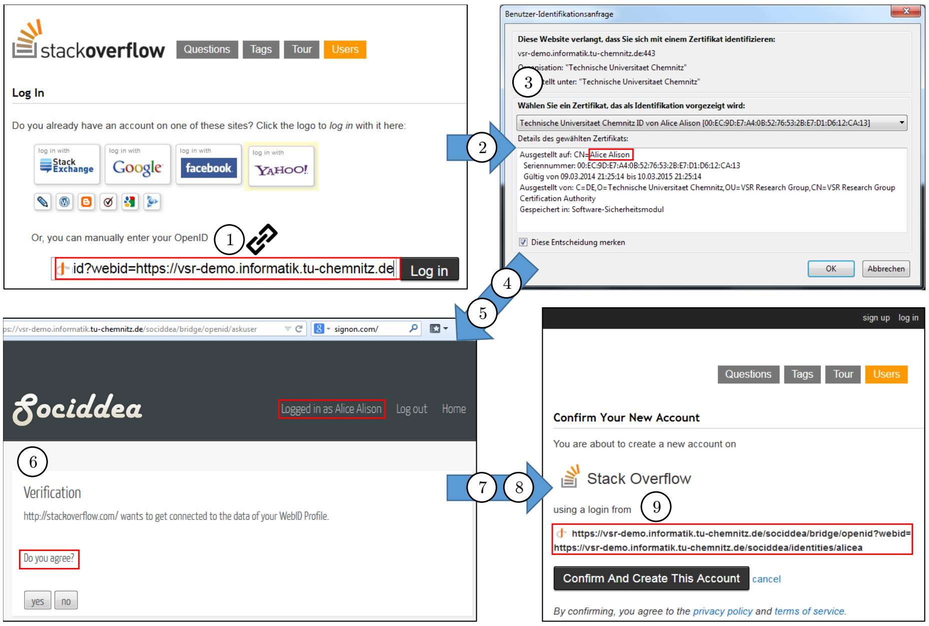


Fig. 1. Sequence Diagram of B3IDS Bridging WebID to OpenID

identifier, B3IDS asks Alice for proving her identity (cf. ③). According to the WebID authentication sequence, Alice has to legitimate herself with the right WebID certificate (cf. ④). To ban an illegitimate WebID URI, B3IDS verifies that her WebID certificate contains a URI that matches the WebID URI in the identifier (cf. ⑤). When Alice grants access to her identity data (cf. ⑥), B3IDS transfers requested identity data to the relying party (cf. ⑦) by redirecting her user agent back to it (cf. ⑧). Alice is logged in successfully (cf. ⑨).

To show B3IDS in practice, we prototypically implemented the approach in the Sociddea WebID provider known from previous work [7]. Fig. 2 presents a usage scenario of logging into the OpenID-enabled Stack Overflow using a WebID identity. The numbers in this figure correlates to the number in Fig. 1. While the implementation can only bridge between WebID and OpenID at the moment, our approach is neither restricted to this combination nor to Sociddea. In the editing box shown in Fig. 2 users can type their OpenID identifier to create a WebID identity based on their OpenID identity. To log into an OpenID relying party users have to use the following identifier: `https://vsr-demo.informatik.tu-chemnitz.de/sociddea/bridge/openid?webid=<WebID-URI>` .

Demonstration. For a live demo and further information about B3IDS and Sociddea visit: `http://vsr.informatik.tu-chemnitz.de/demo/sociddea/`



Do you already have an account on one of these sites? Click the logo to *log in* with it here:

Or, you can manually enter your OpenID

id?webid=https://vsr-demo.informatik.tu-chemnitz.de

Diese Website verlangt, dass Sie sich mit einem Zertifikat identifizieren:

vsr-demo.informatik.tu-chemnitz.de:443

Organisation: "Technische Universitaet Chemnitz"

Ausgestellt unter: "Technische Universitaet Chemnitz"

Wählen Sie ein Zertifikat, das als Identifikation vorgezeigt wird:

Technische Universitaet Chemnitz ID von Alice Alison [00:EC:9D:E7:A4:0B:52:76:53:2B:E7:D1:D6:12:CA:13]

Details des gewählten Zertifikats:

Ausgestellt auf: CN=Alice Alison
Seriennummer: 00:EC:9D:E7:A4:0B:52:76:53:2B:E7:D1:D6:12:CA:13
Gültig von 09.03.2014 21:25:14 bis 10.03.2015 21:25:14
Ausgestellt von: C=DE,O=Technische Universitaet Chemnitz,OU=VSR Research Group,CN=VSR Research Group Certification Authority
Gespeichert in: Software-Sicherheitsmodul

[✓] Diese Entscheidung merken

ps://vsr-demo.informatik.tu-chemnitz.de/sociddea/bridge/openid/askuser   signon.com/

Verification

http://stackoverflow.com/ wants to get connected to the data of your WebID Profile.

Do you agree?

Confirm Your New Account

You are about to create a new account on Stack Overflow

using a login from

https://vsr-demo.informatik.tu-chemnitz.de/sociddea/bridge/openid?webid=https://vsr-demo.informatik.tu-chemnitz.de/sociddea/identities/alicea

By confirming, you agree to the privacy policy and terms of service.

Fig. 2. Logging into Stack Overflow using a WebID identity

3 Conclusion and Future Work

By enabling users to both create WebID identities based on the personal data stored at their OpenID identity providers and use their new WebIDs to log into OpenID relying parties, B3IDS demonstrated the identity bridging concept as an example. In B3IDS, WebID users can authenticate to relying parties using a typical OpenID without impairing their normal user experience. Thus, we expect to increase the user acceptance of WebID. Future work will focus on addressing the security concerns related to B3IDS acting as a mediator between identity concepts with the need of having access to associated user data. We also plan to support further identity concepts like OpenID Connect, which obsoletes OpenID [5]. In addition to this, we will work on retrieving protected profile data taking the user's privacy and secrecy preferences into account.

References

1. Dhamija, R., Dusseault, L.: The Seven Flaws of Identity Management: Usability and security Challenges. IEEE Security & Privacy 6(2), 24–29 (2008)
2. Diodati, M.: Identity Bridges: Uniting Users and Applications Across the Hybrid Cloud (2012), `https://www.gartner.com/doc/2008315/identity-bridges-uniting-users-applications` (accessed March 12, 2014)
3. Inkster, T., et al.: WebID TLS - W3C Editors Draft (2014), `http://www.w3.org/2005/Incubator/webid/spec/tls/` (accessed March 12, 2014)

4. Pettey, C., et al.: Gartner Identifies Six Trends That Will Drive the Evolution of Identity and Access Management and Privacy Management (2012), http://www.gartner.com/newsroom/id/1909714 (accessed March 12, 2014)
5. Sakimura, N., et al.: OpenID Connect Core 1.0 (2014), http://openid.net/specs/openid-connect-core-1_0.html (accessed March 11, 2014)
6. Simmons, H.: ForgeRock Releases Breakthrough Identity Bridge (2013), http://www.reuters.com/article/2013/07/30/ca-forgerock-idUSnBw305380a+100+BSW20130730 (accessed March 12, 2014)
7. Wild, S., Chudnovskyy, O., Heil, S., Gaedke, M.: Protecting User Profile Data in WebID-Based Social Networks Through Fine-Grained Filtering. In: Sheng, Q.Z., Kjeldskov, J. (eds.) ICWE Workshops 2013. LNCS, vol. 8295, pp. 269–280. Springer, Heidelberg (2013)

Cross-Browser Testing in Browserbite

Tõnis Saar[1], Marlon Dumas[2], Marti Kaljuve[1], and Nataliia Semenenko[2]

[1] Software Technology and Applications Competence Center, Estonia
{tonis.saar,marti.kaljuve}@stacc.ee
[2] University of Tartu, Estonia
{marlon.dumas,nataliia}@ut.ee

Abstract. Cross-browser compatibility testing aims at verifying that a web page is rendered as intended by its developers across multiple browsers and platforms. Browserbite is a tool for cross-browser testing based on comparison of screenshots with the aim of identifying differences that a user may perceive as incompatibilities. Browserbite is based on segmentation and image comparison techniques adapted from the field of computer vision. The key idea is to first extract web page regions via segmentation and then to match and compare these regions pairwise based on geometry and pixel density distribution. Additional accuracy is achieved by post-processing the output of the region comparison step via supervised machine learning techniques. In this way, compatibility checking is performed based purely on screenshots rather than relying on the Document Object Model (DOM), an alternative that often leads to missed incompatibilities. Detected incompatibilities in Browserbite are overlaid on top of screenshots in order to assist users during cross-browser testing.

Keywords: Cross-browser compatibility testing, image processing.

1 Introduction

Cross-browser (compatibility) testing aims at finding incompatibilities in the way a Web page is rendered across different combinations of a browser, a browser setting, an operating system (OS) and a hardware platform (herein called a *configuration*). The exact meaning of the term "incompatibility" varies from one testing subject to another and hence cross-browser testing has to take into account the sensitivity of the intended user(s). Incompatibilities may range from missing buttons, to misaligned text blocks, broken images or misplaced elements. In the absence of tool support for cross-browser testing, testers have to open web pages manually and check for differences. This procedure is time-consuming, monotonous and non-scalable given the growing number of configurations that need to be supported by Web applications.

Existing automated methods for cross-browser testing are generally based on an analysis of the Document Object Model (DOM) [1][2][3]. However, the fact that a Web page has very similar DOM structure and parameters across different configurations does not guarantee absence of incompatibilities, as rendering engines may display similar DOMs in rather different ways. Thus DOM-based cross-browser testing techniques suffer from lower recall (high number of missed incompatibilities). Some techniques such as WebDiff [1] apply DOM-based web page segmentation in conjunction with image comparison over pairs of matching segments. But while the latter step improves recall, the DOM segmentation step may still hide incompatibilities.

In contrast to the above techniques, Browserbite employs image processing both for web page segmentation and segment comparison. Specifically, Browsebite combines an image segmentation technique based on detection of discontinuities and colour changes with an image comparison technique based on a combination of geometric features and histograms of pixel intensity distribution. These techniques are complemented by supervised machine learning, so as to take into account user sensitivity.

2 System Overview

Browserbite consists of three main components: screenshot generation, image segmentation and comparison, and classification, as shown in Fig. 1. These components are triggered sequentially when a user inserts a URL of a *web page under test* in Browserbite's interface. The URL is added to a queuing system implemented using Ruby Resque. Different Ruby workers then take specific tasks from the queue and perform the task in question, incl. generating screenshot, resizing image, comparing pair of images, or filtering potential incompatibility via a classification model.

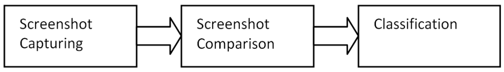


Fig. 1. Overview of Browserbite's tool chain

In addition to a URL, the user specifies a *baseline configuration*, that is a browser-OS-platform for which the user has verified correct rendering. On average, results are displayed in 30 – 45 seconds (with initial results shown incrementally). Detected incompatibilities are highlighted on top of the baseline configuration as shown in Fig 2.

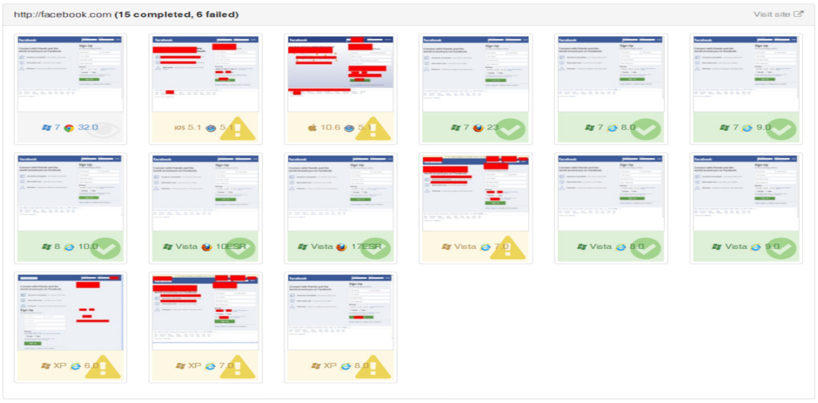


Fig. 2. Example of Browserbite report

2.1 Screenshot Capturing

A number of virtual machines are used to generate screenshots on different browsers and operating systems. Browserbite supports six OS (Windows XP, Vista, 7, 8, Apple OS X 10.6, iOS 6.0) and two browsers with default settings (Google Chrome, Firefox, IE and Safari). A screenshot is generated using the Selenium WebDriver library.

After a web page is loaded, the browser window is maximized. Then the whole window (a.k.a. viewport) is saved as an image. In case of OS X a full-page screenshot has to be composed out of fragments, in a process of scrolling, screenshotting and re-stitching. In Windows, a full-page screenshot can be taken in a single step [4].

2.2 Segmentation and Comparison

In this stage, pairs of screenshot images are compared to find significant differences. One of the images is a baseline image and the other is an image under test.

As mentioned earlier, pixel-by-pixel comparison leads to excessive false positives. Indeed, small misalignments of even one pixel may cause pixel-by-pixel comparison to immediately fail. Accordingly, Browserbite adopts a two-step comparison process. First, images are segmented into smaller so-called regions, which are then matched pairwise. Segmentation helps to prevents false alarms caused by small misalignments of web page elements. The segmented regions can represent for example buttons, forms, headings, text blocks etc. Segmentation in Browserbite is based purely on visual features (discontinuity and colour changes) and is implemented using well-known image processing techniques [5].

In a second step, segments are compared pairwise (one baseline segment versus one image-under-test segment). Pairwise comparison is performed first on geometric features (position and size) and secondly on the values of the histogram of pixel density distribution, following a well-known histogram extraction technique used in computer vision [5]. Each baseline segment is matched to the most similar segment from the image-under-test. If two matched segments have differences in feature parameters beyond a *tolerance threshold*, or if a segment in one image has no matching pair in the other, the segment(s) is/are declared *potentially incompatible*. Tolerance thresholds have been tuned experimentally based on a corpus of images (see below) in such a way as to produce a small number of false negatives (2% of missed incompatibilities, i.e. 98% recall). These thresholds however lead to a precision of 66%. To strike a better tradeoff, Browserbite relies on an additional classification stage.

2.3 Classification

The classification stage is used to classify potential incompatibilities into actual incompatibilities versus false alarms. The classifier uses the same features mentioned above, which are extracted via image comparison (full list of features is given in [6]).

For training and testing the classifier, we used the 140 most popular web pages in Estonia (from the alexa.com list). These web pages were tested manually using Browserbite without the classifier component. As a result 20 000 potential differences were found. From this set 2700 segment pairs were randomly selected. 40 people were asked to classify these 2700 potential incompatibilities into the two classes. As a result 1350 positive and negative cases were obtained.

We tested both decision trees and neural networks (using the OpenCV library) as classification techniques. Neural networks give clearly better results [6]. Plain Browserbite without neural network classifier has precision of 66% and recall 98%. The neural network classifier improves precision to 96% with a recall of 89%, illustrating the trade-offs. Despite this imperfect result, Browserbite's accuracy (F-score) is superior to that of a state-of-the-art tool (Mogotest) [6].

On the background of these trade-offs, Browserbite has been made a commercial product, available on a software-as-a-service basis at: http://www.browserbite.com. It has a growing user base (over 10 000 registered users). At present, Browserbite can produce false positive results while testing pages with dynamic regions (e.g. animations). It is planned to add dynamic region suppression. Dynamic regions are detected by taking a screenshot of a web page with an interval in-between, and comparing the two screenshots using the same technique described above.

3 Conclusion

The Browserbite development experience demonstrates the feasibility and power of cross-browser testing based on image processing. Extensions include the ability to handle Web page flows (as opposed to individual pages) and the adaptation of Browserbite to non-traditional web platforms like smart TV's, billboards and GPS devices.

References

1. Choudhary, S.R., Versee, H., Orso, A.: WEBDIFF: Automated identification of cross-browser issues in web applications. In: 2010 IEEE International Conference on Software Maintenance (ICSM), pp. 1–10 (2010)
2. Choudhary, S.R., Prasad, M.R., Orso, A.: CrossCheck: Combining Crawling and Differencing to Better Detect Cross-browser Incompatibilities in Web Applications. In: 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST), pp. 171–180 (2012)
3. Mesbah, A., Prasad, M.R.: Automated cross-browser compatibility testing. In: Proceedings of the 33rd International Conference on Software Engineering, pp. 561–570 (2011)
4. Kaljuve, M.: Cross-Browser Document Capture System. Master's Thesis, University of Tartu (June 2013), http://tinyurl.com/nlze7ub
5. Shapiro, L.G., Stockman, G.C.: Computer Vision. Prentice Hall (2001)
6. Semenenko, N., Dumas, M., Saar, T.: Browserbite: Accurate Cross-Browser Testing via Machine Learning Over Image Features. In: Proceedings of the 28th International Conference on Software Maintenance (ICSM), pp. 528–531. IEEE Computer Society (2014)

DireWolf Goes Pack Hunting: A Peer-to-Peer Approach for Secure Low Latency Widget Distribution Using WebRTC

István Koren, Jens Bavendiek, and Ralf Klamma

Advanced Community Information Systems (ACIS) Group,
RWTH Aachen University,
Ahornstr. 55, 52056 Aachen, Germany
{lastname}@dbis.rwth-aachen.de
http://dbis.rwth-aachen.de

Abstract. Widget-based Web applications are outperforming monolithic Web applications in terms of distribution of the user interface on many devices and many standard browsers. However, latency of the remote inter-widget communication may be an obstacle for the uptake of Widget-based Web applications in near real-time domains like Web gaming and augmented reality. In this demo paper we show DireWolf 2.0 which is replacing the XMPP server of the DireWolf approach by a client-side relay realized by the means of WebRTC. This is not only decreasing the latency of the distributed interface for any application but also increasing the security by avoiding man-in-the-middle attacks on the XMPP server. This progress is enabling further uptake in Widget-based solutions in advanced Web engineering.

1 Introduction

The Web as a ubiquitous platform allows us to deal with complex tasks within the familiar environment of a Web browser. Mobile devices such as laptops, smartphones and tablets allow us to access Web resources from any possible location. This goes in hand with the shift away from fixed office settings to mobile and dynamic working environments. Yet using multiple devices in parallel is not sufficiently dealt with on the Web. Though the *Responsive Web Design (RWD)* [1] paradigm helps us to open the same Web site on devices with varying screen sizes, it only targets the UI level and does not care about data migration and synchronization issues.

To this end in earlier work we have presented the DireWolf framework [2]. DireWolf is a widget based Web application framework that allows distributing widgets over multiple devices while keeping the application state. It is based on the *Inter-Widget Communication (IWC)* capabilities of the underlying Open Source ROLE SDK[1]. However, the underlying architecture that involves messages being sent over an XMPP server turned out to be an obstacle for latency-sensitive near real-time Web applications.

[1] http://sourceforge.net/projects/role-project/

To solve this problem we hereby introduce the next iteration of the DireWolf framework that uses the recent *Web Real-Time Communication (WebRTC)* draft for sending peer-to-peer messages from browser instances across multiple devices [3]. In the next sections we present our architecture that involves a refurbished message passing and show a preliminary evaluation that confirms an average decrease of message round-trip times of around 80%.

A video of our demo is available at `http://goo.gl/ZV7RJ1`.

2 Peer-to-Peer Distributed User Interfaces

DireWolf 1.0 is using a server-side `Device Manager` that maintains different devices and their respective device profiles e.g. tablet or desktop of a user; the server also stores widget arrangements i.e. which widget is present on which device. Clients initiate the migration of widgets by requesting it at the `Device Manager` which then notifies online devices. The communication between clients and the `Device Manager` as well as between the devices themselves is organized by using publish/subscribe over XMPP. The central routing leads to two major problems: First, the indirection over the server is on the expense of high latency. Second, security is threatened as the XMPP server imposes a single point of failure for man-in-the-middle attacks. Since in most cases where devices are used jointly to master a task they are in immediate vicinity, it is obvious to establish direct peer-to-peer connections for message exchange.

The recent WebRTC draft introduces the DataChannels API that allows Web applications to connect and send arbitrary data to instances running on other devices; thereby firewalls as well as NATs and proxy servers are automatically taken care of. Additionally, WebRTC connections are encrypted by default. Therefore, WebRTC fulfills two major requirements to overcome the aforementioned drawbacks of the existing DireWolf framework: Security as well as low latency through avoiding intermediary servers[2].

To leverage WebRTC we have adapted the architecture of DireWolf. Instead of sending the IWC messages through the XMPP server we introduced a *relay* that acts as a proxy server for all participating devices. We deliberately opted for the relay approach in order to avoid a full-mesh topology of the peer-to-peer network as initial tests showed a performance loss caused by resource-constrained mobile devices. The relaying device now hosts the Message Router that is reponsible for setting up the proxied publish/subscribe node and the WebRTC connections to other clients. We still keep the XMPP connection in place for exchanging endpoint information and the initial connection negotiation process; all other messages are sent over the relay. We have successfully submitted an XMPP extension protocol that describes the DataChannel negotiation process over Jingle to the *XMPP Standards Foundation (XSF)* [4].

[2] In case all firewall-traversal techniques fail WebRTC still redirects the encrypted traffic through *Traversal Using Relays around NAT (TRUN)* servers as last resort.

3 Evaluation

We performed both a comparative technical evaluation as well as a user study to prove our conceptions. For the technical part, we measured the round-trip-times of messages sent from a widget to an instance running on a remote device. Figure 1 shows the results of the test series which included 50 runs. First we measured the round-trip times on the previous DireWolf framework that uses the XMPP server for message routing; the average delay was around 143 ms.

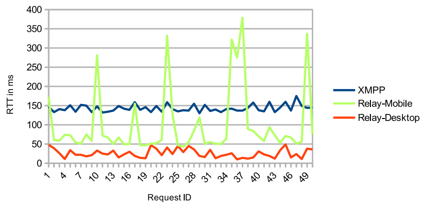


Fig. 1. Latency over various connections

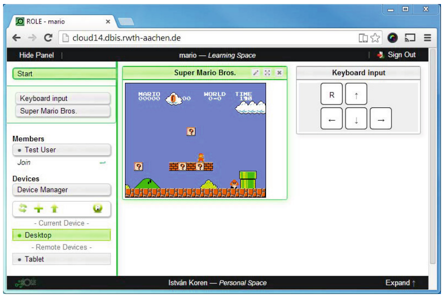


Fig. 2. Evaluation Space

In contrast, test runs performed within the newly developed DireWolf 2.0 framework show a **significant reduction** of round-trip times, however results vary depending on where the computationally intensive publish/subscribe relay was hosted. On a state of the art desktop PC we reached an average round-trip time of around 26 ms. Measurements on a recent smartphone revealed an average time of around 99 ms though there were significant outliers; we assume that these peaks were caused by periodical background tasks occupying the processor.

To the usability end we performed a preliminary evaluation with 16 participants recruited from our research lab. We tested the outcome with two highly latency sensitive applications: near real-time collaborative painting as well as gaming. Both widget spaces were shown on an Android tablet and a Windows laptop side by side and executed first on DireWolf 1.0 and then on DireWolf 2.0. First, users were requested to migrate widgets from one device to another. Then, participants were asked to draw a house on the tablet; the painting was synchronized to a laptop screen via the framework. Finally, users were presented a little platform game that can be seen on Figure 2; the control widgets were placed on the touchscreen device while the interactions were shown on the laptop screen.

For the migration 60% rated DireWolf 2.0 as being "very fast" while around 40% noted it was "fast". DireWolf 1.0 scored around 20% for "very fast" and around 55% for "fast"; around 20% of the interviewed persons even considered the migration time being "slow".

For DireWolf 1.0 the overall synchronization speed was rated as ”slow” by around 40% and even ”very slow” by around 25%, while the proportions for DireWolf 2.0 shifted to ”very fast” or ”fast” by almost 100% of the respondents.

4 Conclusion and Future Work

In this paper we have presented the demo of the DireWolf framework in its newest iteration which allows distributing widget based Web applications to various devices while leveraging the recent WebRTC draft for peer-to-peer style message exchange. For that we have successfully moved the publish/subscribe functionality from the server to a dedicated client in the form of a relay node; client devices connect to the relay for getting updates on the application state.

The technical evaluation has shown that we could decrease the average latency from around 150 ms to around 25 ms with WebRTC relayed by a desktop computer. Furthermore we are now able to prevent man-in-the-middle attacks on the XMPP server for synchronization messages. A usability study verified our findings.

What remains open for future work is a comparison with other DUI or IWC solutions like in the established *Omelette* project. Technical limitations of our systems include the comparatively high initial migration time, as currently widget resources like HTML, JavaScript and image files on a new device still have to be loaded from a Web server. Employing W3C widgets might solve this problem as they are packaged in a single zip file; the system would then send the whole widget bundle over the peer-to-peer connection. Besides, responsifying the widget spaces has a high priority in order to accommodate for today’s huge variation in display sizes and resolutions.

We are committed to tackle these challenges in future versions of the DireWolf framework and are dedicated to continue defining the underlying standards.

Acknowledgements. The work has received funding from the European Commission’s FP7 IP Learning Layers under grant agreement no 318209.

References

1. Nebeling, M., Norrie, M.C.: Responsive Design and Development: Methods, Technologies and Current Issues. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 510–513. Springer, Heidelberg (2013)
2. Kovachev, D., Renzel, D., Nicolaescu, P., Klamma, R.: DireWolf - Distributing and Migrating User Interfaces for Widget-Based Web Applications. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 99–113. Springer, Heidelberg (2013)
3. Burnett, D., Bergkvist, A., Jennings, C., Narayanan, A.: WebRTC 1.0: Real-time Communication Between Browsers (2013)
4. Bavendiek, J.: XEP-0343: Use of DTLS/SCTP in Jingle ICE-UDP Version 0.1, Experimental (2014)

Easing Access for Novice Users in Multi-screen Mashups by Rule-Based Adaption

Philipp Oehme, Fabian Wiedemann, Michael Krug, and Martin Gaedke

Technische Universität Chemnitz, Germany
{firstname.lastname}@informatik.tu-chemnitz.de

Abstract. Novice users often need support to become familiar with a new mashup. The most common problem is that mashups offer a high grade of personalization, such as the user's choice which widgets she wants to use. This problem becomes more difficult in multi-screen mashups, because the user has to decide additionally on which screen the widgets should run. In our recent work we focused on creating multi-screen mashups for enriching multimedia content. That is, a user can watch a video on one screen and also can consume additional content, like a Google Maps excerpt, on another one. This paper presents an approach for rule-based adaption of multi-screen mashups to ease the access for novice users. Therefore, we analyze the users' interaction with the mashup and detect patterns. Based on these patterns we derive rules which will be applied to the mashups of novice users as well as experienced ones. Thus, widgets will be added and arranged automatically on the user's several screens when the execution of a previously generated rule is triggered.

Keywords: Mobile, distributed user interface, distributed displays, multi-screen applications, web applications, mashup, widgets, user interface adaption.

1 Introduction

Mashups allow end users without programming knowledge to easily create applications for their desired target. There, the end users combine small applications, so called widgets, to accomplish their goal. With the emerging amount of mobile devices the opportunities of mashups increase. That is, the mashup is not limited to run on one screen. Rather, multiple screens can be used to create the user interface of one mashup. The following example illustrates the opportunities of these multi-screen mashups: Alice watches a video about an animal documentary on her TV. Meanwhile, she receives additional content on her tablet. This additional content could be a map excerpt about the location of the current scene or the Wikipedia article of the animal which is presented. In this scenario several widgets are described, like a video widget, a map widget or a Wikipedia widget.

One problem for users - especially for novice ones - is that they have to decide which widget they should add. This problem gets more difficult in the field of multi-screen mashups due to the choice on which screen a widget should be placed. Closely related to this topic is the automatic composition in classic single-screen mashups described in [4]. While Roy Chowdhury et al. focus on creating a useful startup configuration of a mashup, they do support neither the adaption of the mashup during runtime nor mashups across multiple screens.

In this paper, we enhance our recent approach for multi-screen mashups to ease the access for novice users to multi-screen mashups by rule-based adaption. By observing and analyzing user actions we create rules which are executed, if a defined action is triggered. For example, Alice adds a video widget to her TV and a map widget to her tablet. This interaction will be transformed into a rule. When Bob adds a video widget to his TV, the rule is triggered and a map widget is automatically added to his tablet.

The rest of this paper is organized as follows: In Section 2 we first present our previous work called SmartComposition and we extend this approach with rule-based adaption in Section 3. Finally, we provide a conclusion and give an outlook on future work in Section 4.

2 SmartComposition Overview

In recent work we described an approach for creating multi-screen mashups called SmartComposition. For illustrating this approach we developed a prototype that enriches a video with related information originating from the Web [1]. In the SmartComposition approach, we describe a workspace as the union of all SmartScreens of one user who uses a mashup. A SmartScreen is an abstract representation of the browser window. It offers the runtime environment for the widgets and also includes functionality to enable the inter widget communication. We differentiate between mobile and desktop SmartScreens that differ in size, accessibility and the available widgets. We also developed a mechanism to enable inter widget communication across multiple screens.

We extend the SmartComposition to offer rule-based adaption of multi-screen mashups for novice users.

3 SmartComposition Enhancements

Our approach enhances the existing architecture of the SmartComposition to effectively support adaption for novice users. The following is a brief description of the extensions we made.

To capture the user actions and also the goal of the users [3, 5] the interaction receiver observes every interaction of the user with the system. We consider actions as every interaction of the user with the system, such as adding new widgets to any SmartScreen as well as moving widgets across several screens or removing them. The captured actions include all required information to reproduce the action by the system, such as the device, the type of the action and other relevant options, such as the widget position. Afterwards, the captured actions are analyzed by the pattern detector to search for patterns on the different

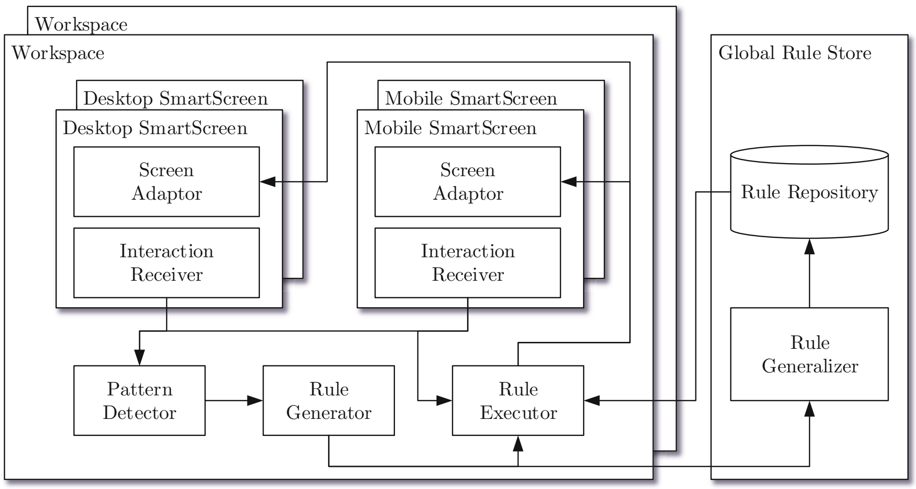


Fig. 1. Architecture of the approach

actions on all devices of a user. That is, the pattern detector examines similar actions which are performed in a specific time interval. Thereby it searches for two actions, one of them is the initiating action and the other one is the resulting action.

If the identified pattern is detected again, the rule generator creates a new rule based on the actions of the pattern. For deciding which pattern will be transformed to a rule we introduce a significance value. This significance value takes two parameters into account: the time difference between the two related actions of one pattern and the amount of occurrences of the same pattern.

The generated rules consist of the initiating action, the resulting action and further parameters to particularize the rule. While the initiating action is triggering the rule later on, the resulting action takes effect in the process of adaption. The rules are utilized to adapt the system to the user [2]. With the aid of the interaction receiver, the rule executor searches for a recurrence of an action that is part of a rule as an initiating action. If an initiating action is found, the rule executor executes the associated resulting action of the related rule. Thus, the action of adding a video widget can lead to the automatic execution of adding a map widget on one of the screens in the workspace.

The rule generalizer uses created user specific rules to derive global rules that are applicable to all users. Therefore, the rule generalizer examines the quality of a rule. This quality is associated to the user's experience in using the mashup. That is, a novice user also creates rules in her own workspace but they are ignored by the rule generalizer because of their minor quality. The generalized rules are saved in the rule repository from which rules can be retrieved from the rule executor in any workspace.

Demonstration. The prototype presented in this paper is available for testing at: `http://vsr.informatik.tu-chemnitz.de/demo/chrooma/adaption/`

4 Lessons Learned and Outlook

In this paper we enhanced our SmartComposition approach to adapt multi-screen mashups by a rule-based system. We observe the users' interaction with the mashup and detect patterns of these interactions. Based on these patterns the system generated rules that follow the abstract template of If-This-Then-That. These rules are executed if a condition is triggered and the multi-screen mashup will be updated. We also introduced a rule-system which differentiates two types. While rules of the first type are applicable only for one specific user, rules of the second type are applicable for all users.

Our future work will focus on a comprehensive user study to examine good thresholds for the rule generation based on the detected patterns as well as the rule generalization. This also includes the time interval within which user interactions will be grouped as one pattern and not as separated ones. We also plan to extend the evolution of generalized rules. That is, rules can be outdated, modified or merged with other rules based on their acceptance of the users.

Acknowledgment. This work was supported by the Sächsische Aufbaubank within the European Social Fund in the Free State of Saxony, Germany (Project Crossmediale Mehrwertdienste für die digitale Mediendistribution).

References

1. Krug, M., Wiedemann, F., Gaedke, M.: Enhancing Media Enrichment by Semantic Extraction. To appear in WWW 2014 Companion: Proceedings of the 23rd International Conference on World Wide Web Companion. International World Wide Web Conferences Steering Committee, Seoul (2014), `http://vsr.informatik.tu-chemnitz.de/publications/2014/002`
2. Muntean, C.H., McManis, J.: Fine grained content-based adaptation mechanism for providing high end-user quality of experience with adaptive hypermedia systems. In: Proceedings of the 15th International Conference on World Wide Web, pp. 53–62. ACM (2006)
3. Rose, D.E., Levinson, D.: Understanding user goals in web search. In: Proceedings of the 13th International Conference on World Wide Web, pp. 13–19. ACM (2004)
4. Roy Chowdhury, S., Chudnovskyy, O., Niederhausen, M., Pietschmann, S., Sharples, P., Daniel, F., Gaedke, M.: Complementary assistance mechanisms for end user mashup composition. In: Proceedings of the 22nd International Conference on World Wide Web Companion, pp. 269–272. International World Wide Web Conferences Steering Committee (2013)
5. Tsandilas, T., Schraefel, M.: Usable adaptive hypermedia systems. New Review of Hypermedia and Multimedia 10(1), 5–29 (2004)

Interactive Scalable Lectures with ASQ

Vasileios Triglianos and Cesare Pautasso

Faculty of Informatics, University of Lugano (USI), Switzerland
{name.surname}@usi.ch
http://asq.inf.usi.ch/

Abstract. Taking full advantage of the Web technology platform during in-class lectures requires a shift from the established scheme of online education delivery that utilizes the video channel to embed all types of content and gathers student feedback via multiple choice questions or textual answers. In this paper we present the design of ASQ to deliver interactive content for use in heterogeneous educational settings with a large number of students, taking advantage of the co-location of students and instructors and building upon the latest capabilities of the Web platform. ASQ is centered around interactive HTML5 presentations coupled with a versatile microformat to create and deliver various types quizzes and scalable, synchronous/asynchronous feedback mechanisms.

1 Introduction

The Web is increasingly used to deliver educational content, being the medium of choice for the popular [1] massive open online courses (MOOCs) and "flipped" (or blended or hybrid) classrooms [2]. Courses are delivered through video lectures and students are assessed either automatically, through multiple choice or text input quizzes, or via peer assessment. Student Response Systems that take advantage of the increased number of Web-enabled devices are finding their way to brick and mortar classrooms replacing traditional hardware clickers. Video as the prominent delivery format of an online lecture is not optimized for the Web medium; it is cumbersome to author and interleave with quizzes and various types of assessment; and it lacks the interactivity and the features that modern Web technologies offer. Today's Web technology can support lectures with highly interactive content such as selectable text, forms, 3D graphics, that can be reactive to a student's input and personalized [3] for different learning styles.

Also of importance are the current assessment models which have remained stale for years: formative or summative assessment is predominantly perfomed through multiple choice or free text quizzes which do not encourage experimentation and creation of original content. In terms of communication models, either between students or between students and teachers, findings suggest that synchronous communication, as a complement to asynchronous communication, can potentially enhance participation in online education [4].

Our goal is to demonstrate ASQ, a platform to create and deliver interactive lectures and gather student feedback in synchronous or asynchronous settings. ASQ makes full use of the HTML5 capabilities of modern Web browsers and provides teachers with the ability to author, deliver and reflect upon the performance of their interactive educational content, while it gives students an additional communication channel to demonstrate their learning progress and actively participate during a traditional lecture.

2 ASQ in the Classroom

ASQ aims to promote the shift from the traditional frontal lecture paradigm (monologue) to interactive bi-directional presentations and discussions (dialogue), through the following features that will be demonstrated: **Delivering educational content:** Interactive lectures are shared with students through a simple URL pointing to the lecture slides. Retrieving the link will connect the student browsers to follow the online presentation. As the instructor navigates through the slides, the navigation events propagate to the connected students which can synchronously follow the material on their Web browsers. This can enhance the accessibility of the lecture, since the material is now displayed in front of the students as it is being explained to them.

Authoring educational content: ASQ manages different kinds of educational content: lecture slides and questions. These can be authored using all the rich multi-hyper-media (images, videos, text, interactive widgets, audio) capabilities of HTML5 compliant Web browsers. Exemplary interactions include selecting or highlighting text, keeping notes for a slide in provided placeholder, playing back audio and video, filling forms, dragging and dropping textual or iconic elements, and any sort of interaction that can be implemented in a Web page using Javascript, CSS3 and HTML5.

Interactive questions and microformat: Questions can be used for both formative and summative assessment, embedded in lectures, collected in quizzes and homework assignements or exams. Each question type is associated with a set of related statistics to be computed over the answers collected by the students. The configuration for rendering questions, assessing answers and visualizing the results of the assessment is controlled with a simple microformat, as shown in the following example:

```html
<!-- a multiple choice quiz with id q-1 -->
<article class="asq-question multi-choice choose-0-n" id="q-1">
  <h3 class="stem"><img src="img/shape.png"></h3>
  <ol>
    <li class="option">This figure is a square.</li>
    <li class="option">This figure is a circle.</li>
    <li class="option">This figure is symmetric.</li>
    <li class="option">This figure has four corners.</li>
  </ol>
</article>
<!-- statistics for the quiz q-1 -->
<article class="stats" data-target-assessment="q-1"
    id="stats-q-1"></article>
```

The microformat parser searches for elements that contain the `asq-question` keyword in their `class` attribute. Once a question is identified the parser searches for its type, in this case `multi-choice`; and for configuration options, in this case students can choose as many of the available multiple choice options they want (`choose-0-n`, but also `choose-1` or `choose-1-n` are possible). Each question type features keywords that provide information about its structure. In the multiple choice example, the class `option` of the `li` elements instruct the parser to store these elements as the options of the multiple choice question. Similarly, statistics are processed with the parser searching for the `asq-stats` keyword. The `data-target-assessment` points to the associated question. Once all questions and statistics are parsed, they are stored in the database and then a markup generator is invoked, that injects necessary markup like form fields and buttons.

Innovative quiz types: Besides standard multiple choice (MC) and single-line text input (STI) quizzes, ASQ currently features two question types specifically targeting the Computer Science domain: code highlight and code input.

Classroom flow: ASQ enhances the educational material by weaving support for complex interactions. Instructors can highlight important points on the presentation and have the marking happen instantly on the students screens. Students can answer quizzes and questions embedded in the slides –individually or in teams– giving instant feedback to the instructor about their level of comprehension. ASQ supplies instructors with a continuous stream of events and statistics related to quizzes, like student progress, correct versus wrong answers, enumeration of actual solutions. Upon receiving the results an instructor may choose to discuss them with individual students, share them with the class through the projector or the students screens to present insightful statistics.

3 Architecture

ASQ follows a client server architecture (Fig. 1). In the backend the logic is implemented in express.js on top of node.js, a choice mandated by the need for efficient I/O operations. The Web server design follows a REST approach treating educational material (lectures, slides, questions, answers) as Web resources.

Bi-directional communication between instructors and students is implemented using WebSockets. This allows for real-time exchange of a fairly big volume of events that are crucial for monitoring progress, supporting complicated assessment modes (for example peer assessment) or fine-grained logging of student actions. User management in ASQ, involves user roles like professors, teaching assistants and students and user permissions defined for each course. Interactions with the website that involve regular HTTP requests use cookie-based authentication, while realtime communication uses token-based authentication and group-based WebSocket namespaces.

The database consists of MongoDB collections for question instances, question types, lectures, sessions, users, and answers. Session events and socket events are stored in a Redis key-value store for scalability and increased performance.

ASQ supports both client- and server-side rendering with Dust.js templates. Question instances are parsed and rendered on the backend, which allows for

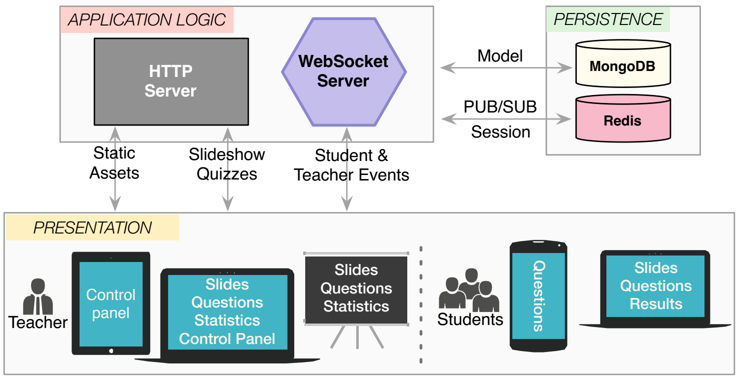


Fig. 1. ASQ Architecture

caching and saves CPU time on the clients. Dynamic elements like graphs and dialog windows are rendered client-side. Moreover, ASQ takes into account display size and user roles to optimize the rendered content. For example, in a student's smartphone with limited HTML5 features and/or a small screen, ASQ only renders the questions so that students can answer them, but does not show the statistics visualizations.

4 Conclusion

In this paper we demonstrate the main features of ASQ, a tool for delivering highly interactive educational content on the Web. Traditionally, instructors may only gather a small sample of answers from the students (who may have to speak up in front of a large audience). ASQ opens a new channel through which teachers can collect the answers of every single student attending a lecture. In the future, we will augment the question delivery system to enable the formation of groups of students to collaboratively solve problems. Finally we plan on evaluating the educational outcomes of ASQ through a summative evaluation that aims to compare a traditional classroom with an ASQ-enabled classroom.

References

1. Shah, D.: MOOCs in 2013: Breaking Down the Numbers. Teasing out trends among the unabated growth of online courses. edSurge (December 2013)
2. Bergmann, J., Sams, A.: Flip your classroom: reach every student in every class every day. International Society for Technology in Education Eugene, OR (2012)
3. Dolog, P., Henze, N., Nejdl, W., Sintek, M.: Personalization in distributed e-learning environments. In: Proceedings of WWW Alt. 2004, pp. 170–179. ACM (2004)
4. Hrastinski, S., Carlsson, S.A.: Participating in synchronous online education. Lund Studies in Informatics (6) (2007)

LiquidML: A Model Based Environment for Developing High Scalable Web Applications

Esteban Robles Luna[1], José Matías Rivero[1,2], and Matias Urbieta[1,2]

[1] LIFIA, Facultad de Informática, UNLP, La Plata, Argentina
{esteban.robles,mrivero,matias.urbieta}@lifia.info.unlp.edu.ar
[2] Also at Conicet

Abstract. The scalability of modern Web applications has become a key aspect for any business in order to support thousands of concurrent users while reducing its computational costs. However, existing model driven web engineering approaches have been focus on building Web applications that satisfy functional requirements while disregarding "technological" aspects such as scalability and performance. As a consequence, the applications derived from these approaches may not scale well and need to be adapted. In this paper we present the LiquidML environment, which allows building Web applications using a model-based approach. In contrast with existing approaches, aspects that help to improve the scalability of a Web application are modeled as first class citizens and as a consequence the applications obtained scale better than its counterparts.

Keywords: Scalability, Model driven development, Web engineering.

1 Introduction

Scalability is the ability of a system to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth [2]. Scalability problems in the applications derived using Model-Driven Web Engineering (MDWE) tools may not appear as soon as they are deployed, but rather after they has been "living" in production for some time. Fixing scalability issues requires a detail diagnosis and consumes many human resources [4] and it has been shown to take 50-70% of the development time [1].

In the Web engineering research area [6], MDWE approaches [5,3,7] have become an attractive solution for building Web applications as they raise the level of abstraction and simplify the Web application development. However, according to [8], little attention has been put to non-functional requirements such as scalability issues as they have been considered a technological aspect that does not need to be modeled. In the same line, the Web applications derived from MDWE approaches pose a rigid/static architecture that cannot be easily changed thus limiting the size of the Web applications that can be built with them. In addition, diagnosing and fixing these problems in production systems (which are the ones that present overload symptoms) becomes cumbersome and impossible to be done in the models thus forcing teams to deal with the generated code losing the high level of abstraction provided by the design models.

In this paper we present *LiquidML*, an environment that helps building high scalable Web applications that can be manipulated at runtime. A LiquidML application can be either derived from MDWE models such as WebML, or it can be built straight inside the environment by manually creating models. The main difference with existing tools is that we do not impose a rigid architecture and those "technological" aspects are modeled; as a consequence we can keep track of their changes and alter the models at runtime in a safe way.

2 LiquidML Models

The LiquidML environment allows engineers to model a set of features that handle Web applications requests as first class citizens. A non-exhaustive list of features includes: logging and profiling, caching, service calls, parallel, asynchronous and sequence execution modes, message queuing and the deployment process. A LiquidML application is a composition of Flows and a Flow describes a sequence of steps that need to happen within a Web request (called *Message* in our approach). A Message has a payload (body) and a list of properties while each step in the Flow is visually identified by an icon and constitutes an *Element* of it. Communication between Elements happens by means of Message interchanges.

To exemplify the concepts, we present a Flow for the product's detail page of an E-commerce web application (Fig. 1). The Element with no incoming arrow represents the Message source listener that will receive incoming requests – in this case, it will receive HTTP request and will transform them in to Messages. The Element connected to the Message source named "Route path" is a ChoiceRouter, which behaves like a choice/switch statement and it will route the message to the "Get info" processor if the request comes to a URL starting with "/product/*". The "Get info" is another router that gets information in parallel from multiple sources. It obtains the product info from the DB: ("Get product info") and triggers the computation of the product's rank, which involves two database (DB) queries ("Get user reputation" and "Get product reviews") and a Processor that computes the rank from this information ("Compute product rank"). Finally, the information gets composed ("Compose data") and used for rendering a Web page in the "Render template" processor.

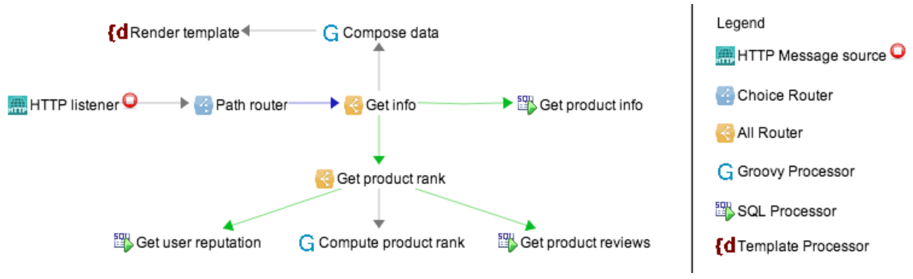


Fig. 1. Product details flow

As aforementioned, to improve the scalability of the Web application some elements suggested in [4] has been modeled as first class citizens. In Fig. 2 we show a modified version of the Product details flow of Fig 1, were some elements for caching purposes have been added to improve the overall scalability of the Web application. In LiquidML, these changes can be applied at design time by adding the elements in the diagram or by means of runtime transformations that change model elements on-the-fly while the application is running.

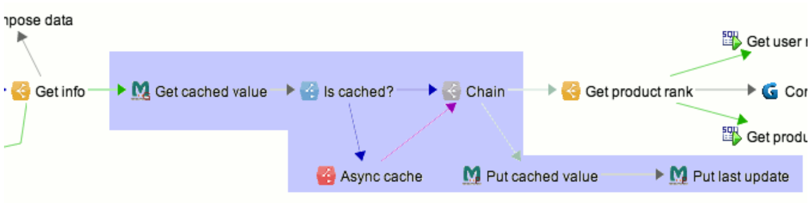


Fig. 2. Product details flow improved with caching elements

3 LiquidML Environment

The LiquidML environment is composed of 2 applications:

- LiquidML editor: it allows defining the applications, modelling Flows and to deploy them to the LiquidML servers. It also, allows developers to apply dynamic transformations to the deployed diagrams in order to deal with the scalability problems and watch for performance and logging information on-the-fly in any environments where the application is running (DEV, QA or even Production).
- LiquidML server: The server is responsible for holding the application definitions, the LiquidML interpreter and notifying the editor about how applications are running. In addition, it regularly checks if it has any pending deployments and if so, it fetches the Application and automatically deploys it.

LiquidML do not impose any rigid implementation architecture. For instance, a complete Web application can be split in 1 front-end app and 2 different service apps, which can be integrated by sending messages to their REST endpoints. In addition, each application, e.g. the front-end app, can be instantiated in multiple servers and combined by load balancers to distribute the load and improve the overall scalability of the application. As an example, in Fig. 3 we present the deployment view of a diagram, which allows us to use the diagnostic tools of LiquidML such as profiling and logging and it also supports performing runtime transformations oriented to improve the scalability of the Web application.

Both, the editor and the server have been built using open source technologies of the JEE stack. We have used Spring and Hibernate for basic service and ORM mapping, Spring MVC and Twitter bootstrap for UI and Jersey for the LiquidML API. As part of this development, we have built the CupDraw framework[1] for building Web diagram editors, which is publicly available. For the technical readers, we invite them to visit the LiquidML site http://www.liquidml.com and check the project's source code and the demonstration videos.

[1] `https://github.com/estebanroblesluna/cupDraw`

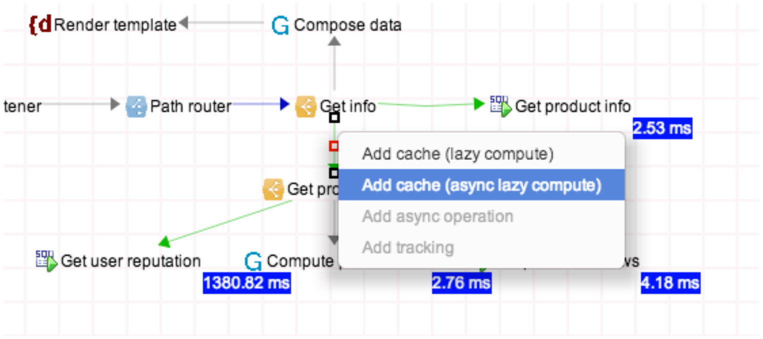


Fig. 3. Deployment view

4 Conclusions

In this demo paper, we have presented the LiquidML environment as a place where Web applications can be either derived from MDWE models or manually created from our "low level" LiquidML models. These models can be then interpreted and dynamically reconfigured at runtime. The environment is web based and it only requires a browser to be used.

References

1. Boehm, B.W.: Software engineering economics. Prentice-Hall, Englewood Cliffs (1981)
2. Bondi, A.: Characteristics of scalability and their impact on performance. In: Proceedings of the 2nd International Workshop on Software and Performance (WOSP 2000), pp. 195–203. ACM, New York (2000)
3. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): A Model-ing Language for Designing Web Sites. Computer Networks and ISDN Systems 33(1-6), 137–157 (2000)
4. Hull, S.: 20 Obstacles to Scalability. Queue 11(7), 20 (2013)
5. Koch, N., Knapp, A., Zhang, G., Baumeister, H.: UML-Based Web Engineering, An Approach Based On Standards. In: Web Engineering, Modelling and Implementing Web Applications, pp. 157–191. Springer, Heidelberg (2008)
6. Rossi, G., Pastor, O., Schwabe, D., Olsina, L.: Web Engineering: Modelling and Implementing Web Applications. Springer (2007)
7. Rossi, G., Schwabe, D.: Modeling and Implementing Web Applications using OOHDM. In: Web Engineering, Modelling and Implementing Web Applications, pp. 109–155. Springer, Heidelberg (2008)
8. Toffetti, G.: Web engineering for cloud computing (web engineering forecast: cloudy with a chance of opportunities). In: Proceedings of the 12th International Conference on Current Trends in Web Engineering (ICWE 2012), pp. 5–19. Springer, Heidelberg (2012)

Managing and Monitoring Elastic Cloud Applications

Demetris Trihinas, Chrystalla Sofokleous, Nicholas Loulloudes, Athanasios Foudoulis, George Pallis, and Marios D. Dikaiakos

Department of Computer Science, University of Cyprus
{trihinas,stalosof,loulloudes.n,afoudo01,
gpallis,mdd}@cs.ucy.ac.cy

Abstract. Next generation Cloud applications present elastic features and rapidly scale their comprised resources. Consequently, managing and monitoring Cloud applications is becoming a challenge. This paper showcases the functionality and novel features of: (i) c-Eclipse, a framework for describing Cloud applications along with their elasticity requirements and deploying them on any IaaS provider; and (ii) JCatascopia, a fully-automated, multi-layer, interoperable Cloud monitoring system. Particularly, we demonstrate how a user can manage the full lifecycle of a three-tier web application and observe, in real-time, how an elasticity management platform automatically scales the application based on various user-defined elasticity requirements, workloads and performance metrics.

1 Introduction

Cloud computing offers organizations the opportunity to reduce IT costs and improve the efficiency of their services. The next generation of Cloud applications present elastic features which allow them to expand or contract their allocated resources in order to meet their exact demands. However, managing and monitoring elastic Cloud applications is not a trivial task. For instance, organizations with large-scale distributed web applications (e.g. online video streaming services) require a deployment comprised of multiple virtual instances, which often have complex inter-dependencies. Despite the fact that current elasticity management platforms such as Amazon Auto Scaling[1] and Rackspace Auto Scale[2] can automatically and seamlessly scale large Cloud applications, these systems are proprietary and limit the application to operate only on specific Cloud platforms. Portability imposes a level of complexity and additional effort, from the Cloud consumer perspective, to *move* an application from one IaaS provider to another. Another downside of the aforementioned systems is that they only handle simplistic boolean requirements based on a limited number of low-level metrics (i.e. CPU, memory usage, etc.) and only support fine-grained elasticity actions (e.g. add/remove virtual instances). Tiramola [1] on the other hand, is an open-source alternative which succeeds in accommodating complex elasticity requirements based on application-level metrics but it is limited to only scale NoSQL databases.

[1] http://aws.amazon.com/autoscaling/
[2] http://www.rackspace.com/cloud/monitoring/

Furthermore, elasticity management requires the monitoring of elastic applications, a challenge that is left unaddressed by many of the existing monitoring tools (i.e. Ganglia[3], Nagios[4]). The complex nature of this task requires for the monitoring system to run in a fully automated manner, detecting configuration changes in the application topology which occur due to elasticity actions (e.g. a new VM is added) or when allocated resource-related parameters change (e.g. new disk attached to VM).

To address the aforementioned challenges which occur when managing and monitoring elastic Cloud applications, we present two powerful open-source tools:

c-Eclipse: A client-side Cloud application management framework which allows developers to describe, deploy and manage the lifecycle of their applications in a clean and graphical manner, under a unified environment. c-Eclipse is built on top of the well-established Eclipse platform. It adheres to two highly desirable Cloud application features: portability and elasticity. Current frameworks such as the ServiceMesh Agility Platform[5] limit users to describe and deploy their applications to a small number of Cloud platforms for which connectors are available and when users want to *move* their application to another Cloud, the description must be altered. Additionally, current frameworks offer limited or no elasticity support by only allowing the definition of fine-grain elasticity requirements. c-Eclipse ensures application portability by adopting the open TOSCA specification[6] for describing the provision, deployment and application re-contextualization across different Cloud platforms. In contrast to other frameworks which also adopt the TOSCA specification [2], c-Eclipse does not require for its users to have any prior knowledge of the TOSCA specification since users describe their application through an intuitive graphical interface which automatically translates the description into TOSCA. Finally, c-Eclipse facilitates the specification of complex, multi-grained elasticity requirements via the SYBL [3] directive language.

JCatascopia [4]: A fully-automated, multi-layer, interoperable Cloud monitoring system. JCatascopia addresses the aforementioned challenges by being able to run in a non-intrusive and transparent manner to any underlying virtualized infrastructure. Current monitoring systems which provide elasticity support [5] [6], require for special entities at the physical level or depend on the current hypervisor to detect topology changes. In contrast to these systems, JCatascopia uses a variation of the publish and subscribe protocol [7] to dynamically detect, at runtime, when monitoring agents have been added/removed from the overall system due to elasticity actions. This is accomplished without any human intervention, special entities or dependence on the underlying hypervisor, allowing users to even monitor applications distributed over multiple Cloud platforms. In addition, JCatascopia provides filtering capabilities to reduce the overhead for metric distribution and storage, and generates high-level application metrics dynamically at runtime by aggregating and grouping low-level metrics.

[3] `http://ganglia.sourceforge.net/`
[4] `http://www.nagios.org/`
[5] `http://www.servicemesh.com`
[6] `http://docs.oasis-open.org/tosca/TOSCA/v1.0/`
[7] Monitoring Agents (metric producers) subscribe to Monitoring Servers instead of the other way around, allowing for them to (dis-)appear dynamically [4].

2 Elasticity Management Platform

In this section we focus on describing the components which comprise an elasticity management platform which incorporates c-Eclipse and JCatascopia.

A developer, at first, uses the c-Eclipse **Application Description Tool** to graphically describe the application topology, software dependencies and elasticity requirements. The graphical description is translated, on the fly, into TOSCA. Then, the developer selects a Cloud provider and via the c-Eclipse **Submission Tool**, the description is submitted to the **Cloud Manager** for deployment. Subsequently, the Cloud Manager parses the portable and platform independent TOSCA description, and initiates the deployment of the Cloud application via the **Orchestrator**. The Orchestrator consists of two sub-components. The first component is the **Cloud Orchestrator**, which is the interface that interacts with the IaaS provider to (de-)allocate the requested Cloud resources. The second component is the **App Orchestrator**, which performs the execution of application specific scripts and ensures the configuration and deployment correctness.

After successfully deploying a Cloud application, users are able, via c-Eclipse, to monitor the deployment, acquire aggregated monitoring metrics from **JCatascopia**, and configure the deployment by refining the elasticity requirements. The **Cloud Manager** (Fig. 1) constantly checks the user-defined elasticity requirements and when a violation is detected, resizing actions are issued. Specifically, the Cloud Manager requests from the **Cloud Orchestrator** to add/remove instances depending on the application demands and from the **App Orchestrator** to configure the application accordingly.

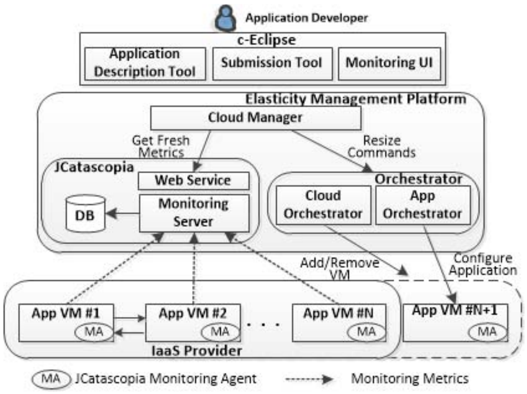


Fig. 1. Elasticity Management Platform

3 Demonstration Description

This demonstration showcases[8] the functionality of the proposed platform by managing the full lifecycle of an elastic Cloud application. Specifically, we will: (i) describe,

[8] Screenshots of c-Eclipse and JCatascopia can be found at: `http://linc.ucy.ac.cy/CELAR/icwe2014`

via c-Eclipse, a Cloud application's topology, software dependencies and relationships between its tiers; (ii) define, via c-Eclipse, elasticity requirements for the elastic components comprising the application; (iii) select a Cloud platform and submit the generated TOSCA description to the Cloud Manager; (iv) monitor both the Cloud resources allocated for the application and its performance by utilizing JCatascopia (Fig. 2); and (v) scale the application, via the Cloud Manager, based on collected metrics and the user-defined elasticity requirements. It must be noted that both the Cloud Manager and Orchestrator are simplistic components developed only to showcase the full potential of c-Eclipse and JCatascopia. Furthermore, attendees may configure the deployment by refining the elasticity requirements. Finally, users will observe real-time graphs for each collected metric, configure monitoring parameters (i.e. sampling rate) and generate graphs by aggregating metrics originated from multiple instances.

Use Case Scenario: we consider a three-tier online video streaming service comprised of: (i) an HAProxy[9] *load balancer* which distributes client requests (i.e. download, upload video) across multiple application servers. (ii) An *application server tier*, where each instance is an Apache Tomcat[10] server containing the video streaming web service. Aggregated tier metrics such as the average *number of connections* and/or *request throughput* can be used to decide when a scaling action should occur; (iii) a Cassandra[11] NoSQL *distributed data storage backend*. Similarly, aggregated metrics such as the average *CPU utilization* and/or *query latency* can be used to scale the Cassandra ring. To stress the video service, we have developed a *workload generator* where the workload form (i.e. sinusoidal, linear), type (i.e. read-heavy, write-heavy, combination) and parameters (i.e. intensity, max execution time) are all configurable.

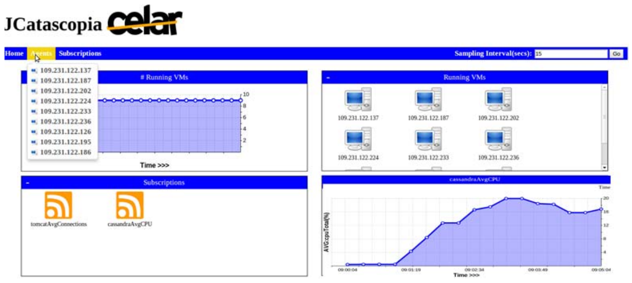


Fig. 2. Screenshot from JCatascopia while running the demo scenario

Acknowledgements. This work was partially supported by the European Commission in terms of the CELAR 317790 FP7 project (FP7-ICT-2011-8).

[9] `http://haproxy.1wt.eu/`
[10] `http://tomcat.apache.org/`
[11] `http://cassandra.apache.org/`

References

1. Tsoumakos, D., Konstantinou, I., Boumpouka, C., Sioutas, S., Koziris, N.: Automated, Elastic Resource Provisioning for NoSQL Clusters Using TIRAMOLA. In: IEEE International Symposium on Cluster Computing and the Grid, pp. 34–41 (2013)
2. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Winery – a modeling tool for tosca-based cloud applications. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 700–704. Springer, Heidelberg (2013)
3. Copil, G., Moldovan, D., Truong, H.L., Dustdar, S.: SYBL: An Extensible Language for Controlling Elasticity in Cloud Applications. In: 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 112–119 (2013)
4. Trihinas, D., Pallis, G., Dikaiakos, M.D.: JCatascopia: Monitoring Elastically Adaptive Applications in the Cloud. In: 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (2014)
5. Clayman, S., Galis, A., Mamatas, L.: Monitoring virtual networks with lattice. In: Network Operations and Management Symposium Workshops (NOMS Wksps), pp. 239–246. IEEE/IFIP (2010)
6. de Carvalho, M.B., Granville, L.Z.: Incorporating virtualization awareness in service monitoring systems. In: IEEE Integrated Network Management, pp. 297–304 (2011)

MAPMOLTY: A Web Tool for Discovering Place Loyalty Based on Mobile Crowdsource Data

Vinicius Monterio de Lira[1], Salvatore Rinzivillo[2], Valeria Cesario Times[1], Chiara Renso[2], and Patricia Tedesco[1]

[1] Universidade Federal de Pernambuco, Recife, Brazil
{vcml,vct,pcart}@cin.ufpe.br
[2] ISTI - CNR, Pisa, Italy
{salvatore.rinzivillo,chiara.renso}@isti.cnr.it

Abstract. Mobility crowdsourced data, like check-ins of the social networks and GPS tracks, are the digital footprints of our lifestyles. This mobility produces an impact on the places that we are visiting, characterizing them by our behavior. In this paper we concentrate on the *loyalty of places*, indicating the regularity of people in visiting a given place for performing an activity. In this demo we show a web tool called MAPMOLTY that, given a dataset of mobility crowdsourced data and a set of Points of Interests (POI), computes a number of quantitative indicators to indicate the loyalty level of each POI and displays them in a map.

Keywords: place loyality, crowdsource mobility data, activities regularity.

1 Introduction

Tracking capabilities of modern geo-based network services provide us with unprecedented opportunities of sensing both movements and activities performed by people. We can exploit these data to monitor and study traffic, animals, maritime and people [3]. Particularly, people produce crowdsourced data from which we can investigate how people use the area where they live. One interesting aspect is to analyze how regularly a person is visiting a given destination. For example, some people tend to go to their preferred restaurant for eating, while some others may tend to explore different restaurants.

We introduce the concept of *loyal user* for a place p and activity A to indicate a user who regularly visits p to perform activity A. For the sake of generality, we associate the Point of Interest's (POI) category to the activity performed in that POI, thus in the rest of the paper we refer to the category or activity as synonyms. From the loyalty measure of the user we can derive a *loyalty map* of a territory. We can discover that some areas have the tendency to be visited by loyal users, while other areas are more characterized by occasional visitors.

The purpose of this demo is to show a web tool called MAPMOLTY (MAPping MObility loyaLTY)[1] that, given a dataset of mobility crowdsourced data, like

[1] http://mapmolty.isti.cnr.it

tracks of individuals, and a set of Point Of Interest, computes a number of measure, called *loyalty indicators*, to summarize the loyalty level of each POI.

The analysis enabled by the tool may be useful in different scenarios. For example, an urban manager may quickly discover attractions that are visited by occasional visitors like tourists or loyal visitors like dwellers. This may be useful in supporting decision making in traffic management and / or urban planning. On the other hand, the loyalty analysis results may be used for marketing purpose by the owners of the POIs to better plan advertising to targeted individuals. The loyalty indicators are also useful for developing services for the citizen like a recommendation system suggesting new destinations according to the observed visitors' behavior.

To the best of our knowledge, this is the first web tool providing a loyalty map of Points of Interests. Other related approaches about individual regularity measures are, for example, in [1, 2, 4]. These papers focus on the individual instead of the places and they measure (using entropy or other measures) how much a user is regular in visiting specific locations. This idea is at the basis of our approach, but here we focus on geographic and activity aspects instead of concentrating on the regularity behavior of a specific user.

2 The MAPMOLTY Tool

The overview of the tool is illustrated in Figure 1. As we can see, it is organized into three components as described below.

Data Collection and Pre-processing. This module organizes the data that is fed into our system. We identify two main sources of data: list of Points of Interest (POI) and set of crowdsourced time stamped visits of individuals to these POIs. There are many POIs dataset available from several web services. To enable the analytical features of our system we require that the **POIs dataset** should provide at least the following attributes for each POI: a unique identifier *poi_id*, spatial coordinates *latitude* and *longitude*, category of the POI *poi_cat* and a name or description of the place. Many POIs collections are organized into a hierarchy,

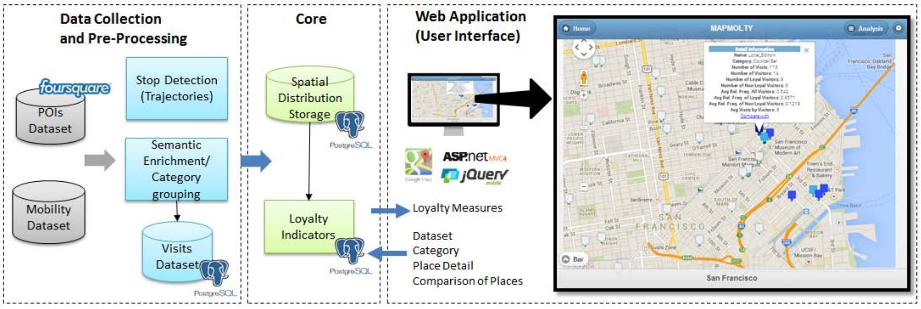


Fig. 1. The overview of MAPMOLTY with the three components

where places with similar categories are grouped under the same larger category. For example, places tagged as *Indian Restaurant* and *Fast Food*, may be associated with the super-category *Restaurant*. Since the super-category is generally informally paired with an activity, we are using the two terms as synonyms. In our experiments we used the set of POIs provided by the Foursquare API[2].

The **Mobility dataset** provides the mobility information to associate a person p to a POI *poi_id* she visited. We require that each visit is represented by a tuple $< \mathit{VisitID}, \mathit{UserID}, \mathit{poi\_id}, \mathit{timestamp} >$, where *VisitID* is the visit unique identifier, *UserID* and *poi_id* represent univocally a user and a place, and *timestamp* is the time when *UserID* arrived at *poi_id*. We call such tuple a *visit*. MAPMOLTY works with many different sources like: Location Based Mobile Social Networks (e.g. Foursquare, Jiepang, Brightkite) and GPS traces. However, a transformation may be needed to convert the mobility data into this format. For example, let us consider a sequence of time stamped GPS points for an individual (called *trajectory*). A trajectory can be transformed into a sequence of visits in a two-step process: *(1)* we detect *stops*, i.e. subsequences of points where the user stands still for a minimum amount of time; *(2)* we associate each stop to the closest POIs provided by the POIs dataset [3]. For the stop detection, we use two parameters: δ, a spatial tolerance threshold and τ, a temporal tolerance threshold. In our experiments, we used $\delta = 50m, \tau = 20min$, meaning that we detect a stop if the user stays in an area of radius $50m$ for at least 20 minutes.

MAPMOLTY uses PostGres SQL 9.3 [4] with PostGIS 2.1.1[5] as Data Base Management System (DBMS) to store the data. PostGIS provides spatial extension for the PostgreSQL database, allowing storage and query of geographical data.

Core. This module analyzes the visits dataset to derive loyalty indicators about the POIs. A visitor is *loyal* to a place when her visits to that place, for performing an activity, are regular. We measure the regularity for a visitor computing his spatial distribution over the frequency of visits to places for a specific activity. Due to lack of space we omit here the mathematical background of this computation. Starting from the spatial distributions from the visitors, MAPMOLTY computes a set of loyalty indicators for each POI: *Number of Visits*; *Number of Visitors*; *Number of Loyal Visitors*; *Number of Non Loyal Visitors*; *Average Relative Frequency of All Visitors*; *Average Relative Frequency of Loyal Visitors*; *Average Relative Frequency of Non Loyal Visitors*; *Average Visits by Visitors*. These indicators show different aspects of the loyalty and are implemented as SQL Procedures developed in PostGres SQL.

The User Interface. The user interface is implemented as a web application, where the user can interact with the map and visualize the information computed from the core component. This tool has been developed using the ASP

[2] `https://developer.foursquare.com/`

[3] More sophisticated Stop-POI association techniques can be used.

[4] `http://www.postgresql.org/download/`

[5] `http://postgis.net/install`

MVC 4 framework[6]. This technology has a Model-View-Controller architecture providing an easy separation between the data manipulation (server side) and the interaction of the user with the web application (client side). MAPMOLTY uses JQuery Mobile 1.3.2 with JQuery 1.9.1 [7] to implement the visual widgets used for the visualization for different types of web-browser devices. The web map widget also uses the JavaScript Google Maps API V3[8] for the visualization and interaction with the map.

The analytical process implemented in MAPMOLTY is structured as follows. When the analyst begins the interaction, the system proposes a list of POI datasets from which to choose the area of interest. Once the area has been selected, the user selects the super-category. The system shows a summary of the available indicators on a map. Each POI is indicated in the map by three visualization properties: marker color, circle size and circle opacity. Based on the loyalty indicators and normalization limits selected by the user, MAPMOLTY computes and plots the values of these visualization properties.

Clicking on the marker the user can visualize the detailed information about the place like the name, the super-category, category and all the indicators values. Other interesting features provided by the web interface are: visually comparison two places, filtering places by their categories and search places by their names.

One peculiarity of this web tool is that it provides a first kernel of features and it can be easily extended with new functionalities. For example, we can incorporate new analysis functions in the database like statistics functions (median, mode and so on) over the data and consequently update the web application to display this new information. Another possible extension is towards the mobile environment. Since we are using a component from JQuery Mobile, the tool can also be easily embedded into a smartphone app using a web panel. This can be particularly useful when associated to a recommendation function based on the loyalty.

Acknowledgments. This work was partially supported by EU-FP7-PEOPLE SEEK (295179), EU-FP7-FET DataSim (270833), FP7-SMARTCITIES-2013 PETRA (609042), CNPq (Brazil) grant 246263/2012-1.

References

1. Eagle, N., Pentland, A(S.): Reality mining: Sensing complex social systems. Personal Ubiquitous Comput. 10(4), 255–268 (2006)
2. Qin, S., Verkasalo, H., Mohtaschemi, M., Hartonen, T., Alava, M.: Patterns, entropy, and predictability of human mobility and life. CoRR, abs/1211.3934 (2012)
3. Renso, C., Spaccapietra, S., Zimányi, E.: Mobility Data: Modeling, Management, and Understanding. Cambridge Press (2013)
4. Song, C., Qu, Z., Blumm, N., Barabási, A.-L.: Limits of predictability in human mobility. Science 327(5968), 1018–1021 (2010)

[6] `http://www.asp.net/mvc/mvc4`
[7] `http://jquerymobile.com/download/`
[8] `https://developers.google.com/maps/documentation/javascript/`

Paving the Path to Content-Centric and Device-Agnostic Web Design

Maximilian Speicher

VSR Research Group, Chemnitz University of Technology, 09111 Chemnitz, Germany
`maximilian.speicher@s2013.tu-chemnitz.de`

Abstract. Content-centric and device-agnostic design are crucial parts of modern web design. They are required to cater for the rapidly growing variety of different web-enabled devices and screen resolutions. We review satire site `motherfuckingwebsite.com` as a drastic example for realizing these aspects. Additional enhancements are proposed that pave the path to up-to-date minimalistic web design. A simple example application is described to illustrate the proposed approach.

Keywords: Content-centric Web Design, Device-agnostic Web Design, Responsive Web Design, Web Interfaces.

1 Introduction

Nowadays, web developers are confronted with a growing amount of novel devices. Thus, also an increasing range of display resolutions has to be addressed (Fig. 1). When the first web-enabled smartphones became popular, it was common practice to provide separately designed versions of the same website. Yet, this approach is highly inefficient considering the range of devices and display resolutions. This calls for the application of responsive web design, i.e., a website flexibly reacts to the device it is accessed with [4]. The usual approach is to combine a fluid grid layout (cf. frameworks like *Bootstrap*[1]) with CSS3 media queries (i.e., breakpoints) to select rules based on the detected device context [4].

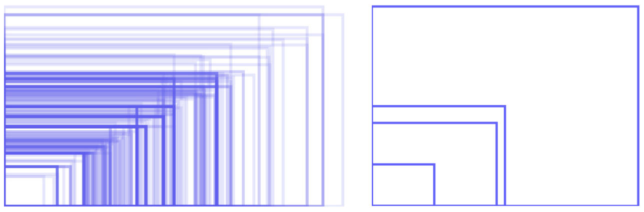


Fig. 1. Comparison of display resolutions of Android devices (left) and Apple devices (right). The graphics have been taken from [5].

[1] `http://getbootstrap.com/` (2014-03-17).

Besides, websites that are overloaded with, e.g., JavaScript libraries or extensive images, can cause problems with web-enabled mobile devices due to slow loading times. Thus, besides responsiveness, modern web design should strongly focus on delivering content while minimizing user distraction through trivialities. A promising method for realizing this is *mobile first* design [6] since the capabilities of mobile devices are still limited compared to desktop computers.

Still, many websites are developed with only three kinds of devices in mind, i.e., desktop, tablet and smartphone, while neglecting resolutions in between.[2] Moreover, cutting-edge websites—although often mobile-ready—tend to focus on a complex visual appearance that is often graphic- and animation-heavy. In this paper[3], we present a review of `motherfuckingwebsite.com` (MFW), which denounces these grievances and satirically claims that it is perfect by following the simplest possible approach to web design. That is, the website is radically content-centric and device-agnostic *without using any device- or resolution-specific breakpoints*. We find that, based on established findings from user experience design, MFW would require adjustments to three particular aspects to provide a more perfect user experience. Yet, although intended to be satire, it is a valid step into the direction of up-to-date minimalistic web design.

2 Related Work

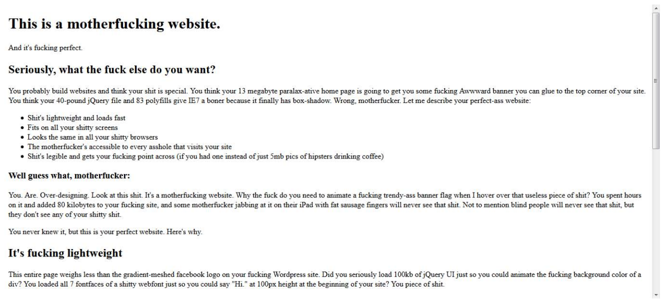
Two well-known strategies for providing websites for different devices are *progressive enhancement* and *graceful degradation* [4]. These focus on starting from one end of the spectrum of devices and then adding more or less sophisticated variants of the website to cater for other devices [4]. The above depend on specific devices while *responsive design* is more oriented towards device-agnosticism. Still, the majority of responsive approaches depend on breakpoints for devices and/or resolutions. The *Goldilocks* approach [1] strongly focuses on text presentation and using as little breakpoints as possible. Conceptually, it is closer to MFW than other responsive approaches such as Bootstrap.

3 Review of motherfuckingwebsite.com

MFW is a website following a drastically minimalistic approach to web design (Fig. 2). Although the site is intended to be satire and highly exaggerated, we believe its concept is a valid step into the direction of content-centric and device-agnostic web design. This was underpinned by numerous positive reactions by users on different social media platforms. We provide a review of MFW w.r.t. established findings from user experience design. Particularly, the website makes the following points to underpin its initial statement that "it's [...] perfect":

[2] `http://www.webdesignerdepot.com/2013/05/common-misconceptions-about-responsive-design/` (2014-03-17).

[3] This paper is based on an earlier blog post by the author, see `http://wp.me/p4gilw-I` (2014-03-17).



This is a motherfucking website.

And it's fucking perfect.

Seriously, what the fuck else do you want?

You probably build websites and think your shit is special. You think your 13 megabyte paralax-ative home page is going to get you some fucking Awwward banner you can glue to the top corner of your site. You think your 40-pound jQuery file and 83 polyfills give IE7 a boner because it finally has box-shadow. Wrong, motherfucker. Let me describe your perfect-ass website:

- Shit's lightweight and loads fast
- Fits on all your shitty screens
- Looks the same in all your shitty browsers
- The motherfucker's accessible to every asshole that visits your site
- Shit's legible and gets your fucking point across (if you had one instead of just 5mb pics of hipsters drinking coffee)

Well guess what, motherfucker:

You. Are. Over-designing. Look at this shit. It's a motherfucking website. Why the fuck do you need to animate a fucking trendy-ass banner flag when I hover over that useless piece of shit? You spent hours on it and added 80 kilobytes to your fucking site, and some motherfucker jabbing at it on their iPad with fat sausage fingers will never see that shit. Not to mention blind people will never see that shit, but they don't see any of your shitty shit.

You never knew it, but this is your perfect website. Here's why.

It's fucking lightweight

This entire page weighs less than the gradient-meshed facebook logo on your fucking Wordpress site. Did you seriously load 100kb of jQuery UI just so you could animate the fucking background color of a div? You loaded all 7 fontfaces of a shitty webfont just so you could say "Hi." at 100px height at the beginning of your site? You piece of shit.

Fig. 2. Screenshot of `motherfuckingwebsite.com`.

1. It is lightweight and loads fast by omitting client-side scripts and graphics.
2. It is completely device-agnostic, i.e., "[the] site doesn't care if you're on an iMac or a [...] Tamagotchi". In fact, it omits any JavaScript- or CSS-based breakpoints and only makes use of the HTML viewport meta tag[4].
3. It is cross-browser compatible.
4. It is accessible for all users, particularly visually impaired ones.
5. It delivers content instead of overloading the site with trivialities.
6. It uses HTML5 tags to leverage semantics.

The above points are correct and necessary for modern web design. Yet, MFW's statement about perfectness, if not meant satirically, would not be completely correct. This is because the site's design follows a rather functional point of view while neglecting important aspects that affect user experience. In this context, we assume that a *perfect user experience*—independent of devices and resolutions—is what makes a website perfect. Particularly, MFW would need adjustments w.r.t. the following three points.

Line Width. Text lines on MFW span across the whole width of the viewport. They might exceed the optimal line length of ∼66 characters [1], particularly in large-screen contexts. More optimal than MFW would be to limit text lines to a width of about 30 characters [1]. Further optimization would include a multicolumn layout[5] and pagination for large screens [3].

Navigation. MFW omits statements about navigation. However, a website featuring larger amounts of content requires corresponding means. Optimally, developers should use a navigation bar fixed to the top of the viewport, which currently is common practice in web design (cf. *Ecosia*[6]). Following the approach of device-agnostic design, the navigation bar must react flexibly if the contained links would span more than one line on small screens.

Visual Aesthetics. Although technically flawless, MFW does not pay attention to site aesthetics, which strongly affect user satisfaction [2] and thus also user experience. Better aesthetics can be realized by using more sophisticated

[4] `https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag` (2014-03-17).
[5] `http://www.w3.org/TR/css3-multicol/` (2014-03-17).
[6] `http://www.ecosia.org/what` (2014-03-17).

typography and color schemes, e.g., for better contrast. This does not require the use of extensive graphics, which is a major aspect of MFW.

The above points underpin that MFW cannot be a completely *perfect* approach to web design. Yet, they also show that only little changes to the site can make for a serious approach to up-to-date minimalistic web design. Given the increasing diversity of web-enabled devices, we believe that more content-centric and device-agnostic design—as promoted by MFW—are an integral part of the future of web engineering.

Example Application. We have designed a website featuring a fixed navigation bar at the top of the viewport. Established approaches like Bootstrap engage CSS3 media queries to determine whether the standard navigation has to be adapted. Since it is cumbersome to cover all potential resolutions (Fig. 1) using media queries, we leverage a more device-agnostic approach. A script adds a second navigation bar featuring only a single dummy link to the site. This additional navigation is placed outside the viewport. On each window resize event, we compare the height of this navigation bar with the real navigation. If the latter has a greater height than the hidden navigation, we know that the contained links span more than one line and should be adjusted to form a dropdown menu. This approach is in accordance with MFW and the proposed enhancements. A corresponding demo is available at `http://www.maximilianspeicher.tk/DAD/`.

4 Conclusion

The increased proliferation of novel devices and growing number of screen resolutions makes it difficult to design websites that can react flexibly to all of them. This requires highly device-agnostic approaches to web design. Also, focusing on content more strongly supports modern, device-independent websites. We provide a review of `motherfuckingwebsite.com`, which is a minimalistic satire site drastically addressing current problems in (cross-device) web design. Additional enhancements to this website make for a valid and feasible approach to up-to-date content-centric and device-agnostic webdesign.

References

1. Armstrong, C.: The Goldilocks Approach to Responsive Design, `http://goldilocksapproach.com/article/`
2. Lavie, T., Tractinsky, N.: Assessing dimensions of perceived visual aesthetics of websites. IJHCS 60(3) (2004)
3. Nebeling, M., Matulic, F., Streit, L., Norrie, M.C.: Adaptive Layout Template for Effective Web Content Presentation in Large-Screen Contexts. In: Proc. DocEng. (2011)
4. Nebeling, M., Norrie, M.C.: Responsive Design and Development: Methods, Technologies and Current Issues. In: Proc. ICWE (Tutorials) (2013)
5. OpenSignal: Android Fragmentation Visualized, `http://opensignal.com/reports/fragmentation.php`
6. Wroblewski, L.: Mobile First. A Book Apart (2011)

Twiagle: A Tool for Engineering Applications Based on Instant Messaging over Twitter

Ángel Mora Segura, Juan de Lara, and Jesús Sánchez Cuadrado

Universidad Autónoma de Madrid, Spain
{Angel.MoraS,Juan.deLara,Jesus.Sanchez.Cuadrado}@uam.es

Abstract. Microblogging services, like Twitter, are widely used for all kind of purposes, like organizing meetings, gathering preferences among friends, or contact community managers of companies or services.

With suitable automation, tweets can be used as a dialogue mechanism between users and computer applications, and we have built a tool, named Twiagle, to construct tweet-based applications. Twiagle includes a pattern-matching language to express the interesting parts to be detected and selected from tweets, and an action language to query matched tweets, aggregate information from them or synthesize messages.

1 Introduction

Microblogging and instant messaging systems are booming nowadays, thanks in part to the proliferation of smartphones and mobile devices. Services like Twitter[1] or WhatsApp[2] are extremely used nowadays to connect with friends, or to organize social activities. These services are not only used for leisure, but most companies and brands use these services to keep in contact with clients.

In this setting, we observe a growing need to automate social activities, leveraging on popular social network platforms, like Twitter. On the one hand, users of social networks – possibly lacking any programming skills – may wish to define simple applications involving the participation of a community of users. On the other, companies may like to open their information systems to social networks platforms, but this integration effort needs to be done by hand.

We claim that social networks based on instant messaging, in particular Twitter, are suitable as front-ends for computer-based applications. We call them *tweet-based* applications, and they present many advantages in some scenarios. First, instant messaging systems are designed to support a high load of users and messages, serving as a robust front-end, difficult to achieve for companies or end users. Second, many people are familiar with Twitter, and have it already installed. Hence, they do not need to learn a new application, or even install a new one. Third, applications can leverage from Twitter's social network structure.

We foresee three main kinds of scenarios for tweet-based applications. In the first one, Twitter is used as a front-end, which then needs to be connected to an

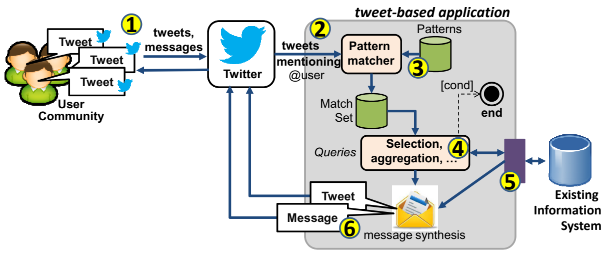
[1] http://www.twitter.com
[2] http://www.whatsapp.com/

existing information system (e.g., an airport may send notifications with flight information, or with status updates via Twitter to interested users). In the second scenario, small, simple, self-contained applications can be designed by unexperienced end users (e.g., outdoors educational games based on quizzes). Finally, an important scenario is the quick construction of applications to coordinate a large amount of people upon unexpected events, like natural disasters.

These scenarios present several challenges for this technology. First, if tweets are used as simple communication mechanism with the application, the relevant information needs to be extracted from them. Second, a mechanism is needed to specify simple actions, like querying the extracted information, or synthesizing messages. Finally, a quick, easy way for constructing this kind of applications is needed, enabling their use by non-experts, but supporting also their deployment into servers, and their integration with existing information systems.

This paper presents *Twiagle*, a tool for constructing tweet-based applications, including a Domain Specific Language (DSL) for expressing patterns, and a DSLs for describing actions. Sec. 2 describes our architecture for *Tweet-based applications*, Sec. 3 describes *Twiagle* using an example, and Sec. 4 ends with the conclusions and prospects for future work.

2 Architecture



The working scheme of our solution for tweet-based applications is shown in the inset figure, where the numbers illustrate a typical interaction. Firstly (label 1), users send tweets or private messages via Twitter. Then, the relevant information in tweets is extracted. Our solution relies on the definition of patterns, expected to be found in tweets. Not every tweet is sought, buy only those mentioning the user associated to the application, or private messages directed to it (label 2). The patterns (label 3) are defined by social media experts, or software engineers. A typical application may include different queries, selecting the relevant concepts in matching tweets, or calculating different aggregation values from them (label 4). In addition, data can be obtained or sent to existing information systems (label 5). The data extracted from queries, or provided by the information system can be used to synthesize tweets or private messages, directed to the users (label 6). Finally, conditions can be defined to signal the end of the execution.

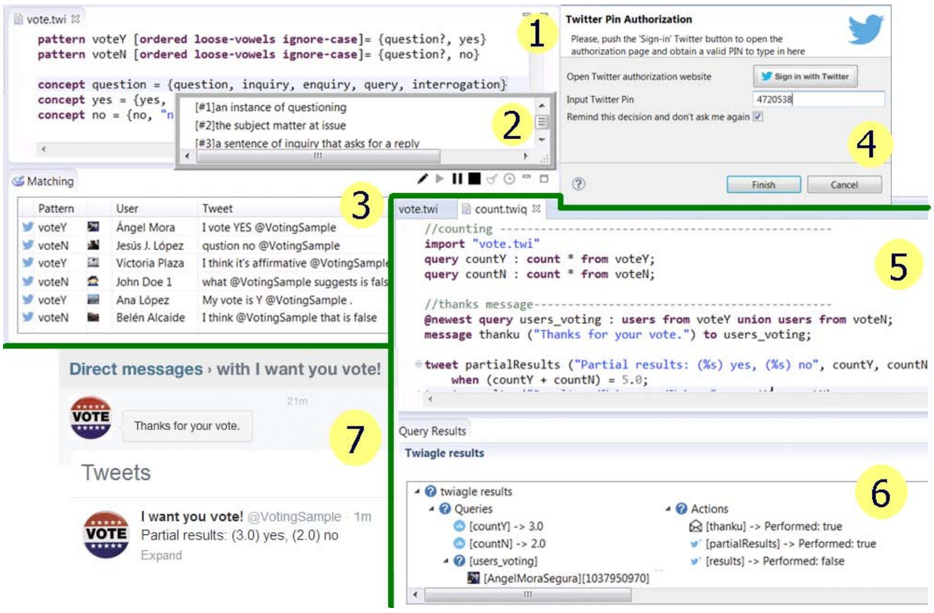
In order to facilitate the construction of such system, we provide a Model-Driven Engineering solution, based on two domain-specific languages (DSLs). The first DSL (called *Twittern*) helps in the definition of relevant patterns, and concepts to be found in them. The latter are sets of relevant words, or fragments, and sets of synonyms can be automatically extracted from Wordnet [??]. The second DSL, called *Twition*, is targeted to the description of the processing logic of tweet-based applications. It allows defining queries on tweets matching some pattern, using an SQL-like syntax. Queries can be used to select relevant information from tweets, or to calculate aggregated information from a set of tweets. The DSL also provides commands to synthesize private messages and tweets. Finally, it is also possible to define *data hooks*, a way to push extracted data into an existing information system, or to gather data from it. The next section describes a tool that realizes this approach.

3 Twiagle by Example

We show *Twiagle*'s capabilities through an example consisting in a simple voting among a set of users (see Fig. 1). The first step is to describe the interesting information in Tweets, using the *Twittern* DSL. A pattern is made of concepts, and in its simplest form, a concept is a set of words, which can be either defined explicitly by the designer, or can be automatically taken from a synonym set provided by Wordnet. We have also included specific Twitter concepts, like patterns to detect user names, URLs (specially pictures), and to define collections of interesting hashtags. The meta-data information present in tweets, like the originator, date or geoposition can be retrieved and does not need to be explicitly declared in patterns. Patterns also indicate if concepts have to appear in some specific order, or allow the interleaving of concepts with other words. It is also possible to specify that some concept cannot occur in a pattern, and whether concepts are to be sought ignoring upper/lower case, accents, and permitting missing vowels, as this is a usual idiom in tweets.

As a second step, our approach considers the description of actions by means of the *Twition* DSL. Twition allows issuing queries using an SQL-like syntax. They may refer to a set of matches of a pattern, as if they formed an SQL table, and the concepts in the pattern, as if they were SQL columns. Three kinds of specialized queries can be issued: Select (to select some concepts from a set of tweets matching a pattern), Adding (to perform some arithmetical operation on result sets), and Metadata queries (to obtain a result set made of some tweet metadata). Similar to data stream management systems [1] we may query using temporal windows. Currently, we support two kinds of temporal windows, one considering all data, and another one with the last tweet (@newest annotation). Once data becomes available from queries, messages can be composed and sent to a collection of users either publicly (command tweet), or in private, directed to a certain user (command message). In addition, received tweets can also be retweeted, and be categorized as favorite. Other commands include facilities to exchange data with an external source, and to signal the application end.

Each action has a name, so that actions can refer to the data they produce simply by that name. The type of data does not need to be declared, but it is inferred by simple rules. The execution model of Twition is based on data flow, relying on data dependencies, the recommended execution model for reactive, event-driven, scalable applications [3]. In this way, an action is performed as soon



```
pattern voteY [ordered loose-vowels ignore-case]= {question?, yes}
pattern voteN [ordered loose-vowels ignore-case]= {question?, no}

concept question = {question, inquiry, enquiry, query, interrogation}
concept yes = {yes,
concept no = {no, "n
```

[#1]an instance of questioning
[#2]the subject matter at issue
[#3]a sentence of inquiry that asks for a reply

Pattern	User	Tweet
voteY	Ángel Mora	I vote YES @VotingSample
voteN	Jesús J. López	qustion no @VotingSample
voteY	Victoria Plaza	I think it's affirmative @VotingSample
voteN	John Doe 1	what @VotingSample suggests is fals
voteY	Ana López	My vote is Y @VotingSample .
voteN	Belén Alcaide	I think @VotingSample that is false

Twitter Pin Authorization
Please, push the 'Sign-in' Twitter button to open the authorization page and obtain a valid PIN to type in here
Open Twitter authorization website
Input Twitter Pin: 4720538
Remind this decision and don't ask me again ✓

```
//counting -------------------------------------------
import "vote.twi"
query countY : count * from voteY;
query countN : count * from voteN;

//thanks message---------------------------------------
@newest query users_voting : users from voteY union users from voteN;
message thanku ("Thanks for your vote.") to users_voting;

tweet partialResults ("Partial results: (%s) yes, (%s) no", countY, countN
    when (countY + countN) = 5.0;
```

```
twiagle results
  Queries
    [countY] -> 3.0
    [countN] -> 2.0
    [users_voting]
      [AngelMoraSegura][1037950970]
  Actions
    [thanku] -> Performed: true
    [partialResults] -> Performed: true
    [results] -> Performed: false
```

Direct messages › with I want you vote!
Thanks for your vote. 21m

Tweets
I want you vote! @VotingSample · 1m
Partial results: (3.0) yes, (2.0) no

Fig. 1. (1) Defining patterns with Twittern, (2) Using Wordnet, (3) Testing with live tweets, (4) Authorizing Twiagle to use Twitter account data, (5) Defining actions with Twition, (6) Execution Debug, (7) Results shown in the Twitter console

as its data becomes available, unless it contains an explicit trigger, in which case it is executed when the data is available *and* the trigger becomes true.

Twiagle includes a console to test patterns against live tweets, as well as an execution debug, showing the results of queries and actions performed. The tool is available at `http://www.miso.es/tools/twiagle.html`.

4 Conclusions and Future Work

In this paper, we have introduced *Twiagle*, a tool to build tweet-based applications. We are currently increasing the expressiveness of *Twittern*, improving *Twition* with new primitives, taking inspiration from data-stream systems for tweet querying. We are also working on the deployment mode, and considering support for other social networks, enabling inter-platform applications.

Acknowledgements. This work has been funded by the Spanish Ministry of Economy and Competitivity with project “Go Lite” (TIN2011-24139).

References

1. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: PODS, pp. 1–16. ACM (2002)
2. Miller, G.A.: Wordnet: A lexical database for english. CACM 38(11), 39–41 (1995)
3. Reactive manifesto, `http://www.reactivemanifesto.org`

Using Linked Data for Modeling Secure Distributed Web Applications and Services

Falko Braune, Stefan Wild, and Martin Gaedke

Technische Universität Chemnitz, Germany
{firstname.lastname}@informatik.tu-chemnitz.de

Abstract. The increasing service orientation of today's Web applications enables swift reaction on new customer needs by adjusting, extending or replacing parts of the Web application's architecture. While this allows for an agile response to change, it is inappropriate when it comes to security. Security needs to be treated as a first thought throughout the entire lifecycle of a Web application. The recently proposed WAMplus approach does not only offer an expressive, extensible and easy-to-use way to model a Web application architecture, but also puts a strong emphasis on the security. In this paper we present an exemplary implementation of WAMplus using the Sociddea WebID identity management system known from prior work. There, we show how WebID is used to identify, describe and authenticate Web applications and services while taking their protection through WAC and fine-grained data filters into account.

Keywords: Modeling, Security, Identity, Protection, Linked Data, WebID.

1 Introduction

Modern Web applications are facing changing costumer needs, short time to market, and an increasing degree of distribution. The service-oriented architecture (SOA) design pattern supports developing such Web applications by providing a set of best practices for organizing distributed capabilities. For responding to change, agile methodology fits well in this context. When it comes to security, however, it is inappropriate to apply this approach without sufficient consideration. Security needs not to be treated as an afterthought, but as a first thought throughout the entire lifecycle of a Web application [5]. While there are various tools which support developers in modeling and building Web applications, they do not holistically address the security of the entire Web application architecture [6].

In a recent work, **(author?)** proposed the WAMplus approach [8]. It does not only offer an expressive, extensible and easy-to-use way to model a Web application architecture, but also puts a strong emphasis on the security.

This work describes a prototypical implementation of the WAMplus approach into an existing WebID identity provider and management system. By exemplarily integrating the approach into Sociddea (`http://www.sociddea.com/`), we show its applicability, demonstrate its use in practice and strive to increase its adoption.

The rest of the paper is organized as follows: Sect. 2 discusses related work. Sect. 3 demonstrates the WAMplus approach. Sect. 4 concludes the paper.

2 Related Work

Model-driven Web engineering (MDWE) approaches like OOWS, UWE or WebML aim at providing means for a systematic and efficient engineering of Web applications [3]. Yet, the variety of existing domain-specific and often proprietary model description languages reduces interoperability. An integral and widely adopted solution for addressing the security topic in MDWE is missing. Dealing with the interoperability concern, semantic vocabularies for describing Web services in RDF typically consider only one type of service aspect: SAWSDL or OWL-S for SOAP-based services; SA-REST or ROSM for RESTful services [2]. OpenID or Facebook Connect are widely adopted identity management systems [1], but their limited extensibility makes them inappropriate to directly identify and attach data to Web applications and services. The Access Control Ontology (ACO) semantically specifies role-based protection of URI-identifiable resources via RDF-based access control lists. Compared to WAC, it is not yet widely used [7].

3 Utilizing WAMplus to Model Secure Web Applications

WAMplus enriches the WebComposition Architecture Model (WAM), known from prior work [4], with 1) semantic descriptions, 2) universal identification, and 3) protection through fine-grained access control for Web application and services. The semantic description of Web services allows for dynamic adaption to interface changes or feature updates. The identification facilitates interconnecting components and establishing authentication through WebID. WAMplus suggests to rely on WAC and customized views for protecting resources, also including descriptions of Web applications and services, at different granularity levels [8].

For the *description* of SOAP-based and RESTful Web services and to maintain interoperability, we use the WSDL RDF mapping (`http://www.w3.org/TR/wsdl20-rdf/`) combined with WebID. Being an identity concept, WebID (`http://www.w3.org/2005/Incubator/webid/spec/`) enables *identification* and authentication, and facilitates more detailed description. It consists of three artifacts: The *WebID URI* refers to a subject, like a Web service, and links to a *WebID profile* storing the subject's identity data and public keys using Linked Data. WebID profiles rely on RDF to semantically describe a subject's attributes. The *WebID certificate* is an X.509 certificate that includes a WebID URI identifying the subject. Matching the public key in both WebID certificate and profile enables subject authentication.

To protect resources and descriptions, the WebAccessControl (WAC) (`http://www.w3.org/wiki/WebAccessControl`) is a RDF-based vocabulary that defines access rules for URI-addressable resource. The WebID Profile Filter Language enables specifying fine-grained filters to protect data *within* user profiles [7].

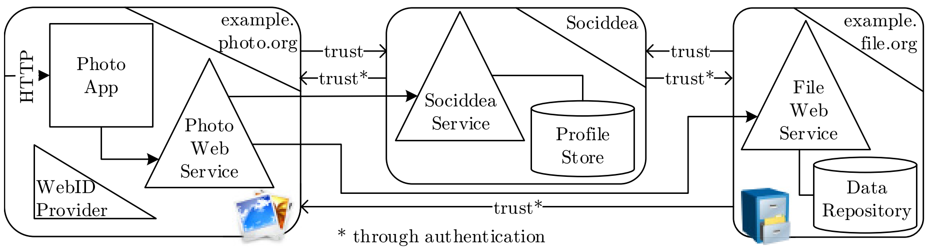
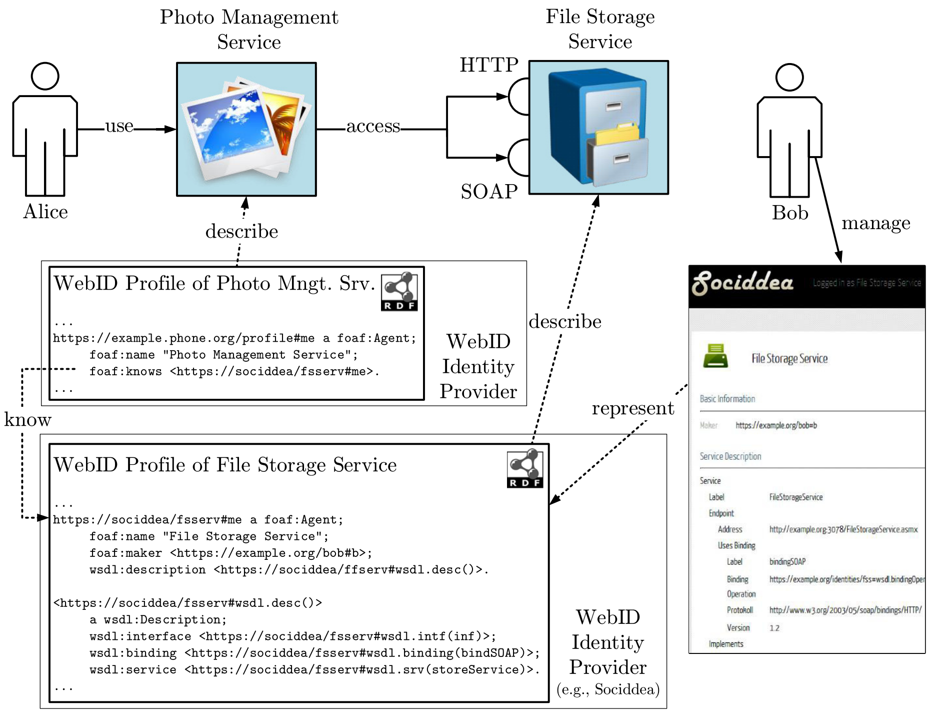


Fig. 1. WAM example

Combining these technologies facilitates a security-focused modeling of Web applications and services that also takes interoperability into account.

To demonstrate WAMplus in practice, we prototypically implemented it in the Sociddea WebID identity provider proposed by **(author?)** in [7]. Yet, WAMplus is not limited to this system. Fig. 1 illustrates an example Web application modeled with WAM. There, a file Web service contains data a photo Web service intents to use. Sociddea is used here to publish Web service descriptions as WebID profiles.



WebID Profile of Photo Mngt. Srv.

```
...
https://example.phone.org/profile#me a foaf:Agent;
     foaf:name "Photo Management Service";
     foaf:knows <https://sociddea/fsserv#me>.
...
```

WebID Profile of File Storage Service

```
...
https://sociddea/fsserv#me a foaf:Agent;
     foaf:name "File Storage Service";
     foaf:maker <https://example.org/bob#b>;
     wsdl:description <https://sociddea/ffserv#wsdl.desc()>.

<https://sociddea/fsserv#wsdl.desc()>
     a wsdl:Description;
     wsdl:interface <https://sociddea/fsserv#wsdl.intf(inf)>;
     wsdl:binding <https://sociddea/fsserv#wsdl.binding(bindSOAP)>;
     wsdl:service <https://sociddea/fsserv#wsdl.srv(storeService)>.
...
```

Sociddea Logged in as File Storage Service

File Storage Service

Basic Information

Maker	https://example.org/bob#b

Service Description

Service	
Label	FileStorageService
Endpoint	
Address	http://example.org:3078/FileStorageService.asmx
Uses Binding	
Label	bindingSOAP
Binding Operation	https://example.org/identities/fss#wsdl.bindingOper
Protokoll	http://www.w3.org/2003/05/soap/bindings/HTTP/
Version	1.2
Implements	

Fig. 2. Usage scenario of photo management system showing WAMplus in practice

As an example, Alice wants to use this photo service to create a new album, as illustrated by Fig. 2. Her photos are stored on the file service, which is operated by Bob. When deploying the file service, Bob also published the service description as a WebID profile using Sociddea. He uses the provided graphical user interface to create a new profile and to add the WSDL RDF description. All following steps are executed during the run-time. Requesting the description of the file service through the photo service invokes the WebID authentication process of Sociddea. In the Sociddea system, every request to a resource has to pass the WAC-type access control rules. Assuming that WAC only allows authenticated users to access profile resources, the photo service has to authenticate with its own WebID. Having access to the profile, its WSDL RDF description provides all necessary information for addressing and binding the service to interact with it. Consistent linking between the resources is established by the use of predicates like *foaf:maker* or *foaf:knows*. If the photo service encounters a problem, Alice or other services could retrieve the operator's contact data by following the WebID URI to Bob's profile being linked to in the WebID profile of the file Web service.

Demonstration. For a live demo and further information about Sociddea and WAMplus visit: `http://vsr.informatik.tu-chemnitz.de/demo/sociddea/`

4 Conclusion

Combining semantic description, universal identification, and access control with WAM's modeling capabilities, WAMplus contributes to designing and managing secure distributed Web applications and services. Our approach assists Web engineers in modeling SOA-based Web applications by creating machine-readable big pictures of their architecture, by using WebID to identify and describe Web services with WSDL+RDF, and by protecting resources with WAC and fine-grained filters. In future work we intend to apply the WAMplus approach in more scenarios to discover patterns relevant to the evolution of Web applications. There we will research the topic of dynamic service replacement and delegation.

References

1. El Maliki, T., et al.: A Survey Of User-centric Identity Management Technologies. In: The International Conference on Emerging Security Information, Systems, and Technologies, SecureWare 2007, pp. 12–17. IEEE (2007)
2. Kim, C.S., et al.: Building semantic ontologies for RESTful web services. In: CISIM, pp. 383–386 (2010)
3. Koch, N., et al.: Model-driven Web Engineering. Upgrade 9(2), 40–45 (2008)
4. Meinecke, J., et al.: Enabling Architecture Changes in Distributed Web-Applications. In: Web Conference, LA-WEB 2007, pp. 92–99. IEEE (2007)
5. Papazoglou, M.P., et al.: Service-Oriented Computing: State of the Art and Research Challenges. IEEE Computer 40(11), 38–45 (2007)

6. Saleem, M.Q., et al.: Model Driven Security Frameworks for Addressing Security Problems of Service Oriented Architecture. In: ITSim, vol. 3, pp. 1341–1346. IEEE (2010)
7. Wild, S., Chudnovskyy, O., Heil, S., Gaedke, M.: Protecting User Profile Data in WebID-Based Social Networks Through Fine-Grained Filtering. In: Sheng, Q.Z., Kjeldskov, J. (eds.) ICWE Workshops 2013. LNCS, vol. 8295, pp. 269–280. Springer, Heidelberg (2013)
8. Wild, S., Gaedke, M.: Utilizing Architecture Models for Secure Distributed Web Applications and Services. Information Technology Special Issue on Architecture of Web Applications (Accepted Journal Paper, Published Q2/2014)

WaPPU: Usability-Based A/B Testing

Maximilian Speicher[1,2], Andreas Both[2], and Martin Gaedke[1]

[1] Chemnitz University of Technology, 09111 Chemnitz, Germany
{maximilian.speicher@s2013,martin.gaedke@informatik}.tu-chemnitz.de
[2] R&D, Unister GmbH, 04109 Leipzig, Germany
{maximilian.speicher,andreas.both}@unister.de

Abstract. Popular split testing approaches to interface optimization mostly do not give insight into users' behavior. Thus, a new concept is required that leverages usability as a target metric for split tests. *WaPPU* is a tool for realizing this concept. It learns models using a minimal questionnaire and can then predict usability quantitatively based on users' interactions. The tool has been used for evaluating a real-world interface. Results underpin the effectiveness and feasibility of our approach.

Keywords: Usability, Metrics, Interaction Tracking.

1 Introduction

Usability is an utterly important factor for successful interfaces. Yet, in today's e-commerce industry, effective methods for determining usability are applied only at very slow iteration cycles (i.e., mainly before a website or redesign goes live). This is because such established usability evaluation methods are costly and time-consuming from a company's point of view. User testing and expert inspections are two of the most prominent examples (cf. [3,6]). Contrary, interfaces are mostly optimized based on *conversions* and more efficient split tests (cf. *Visual Website Optimizer*[1]) during operation. A conversion is a predefined action completed by the user, e.g., a submitted registration form. In a common split test, the interface version which generated the most conversions is considered best. However, this gives no insights into the user's actual behavior.[2]. In fact, suboptimal interfaces can lead to accidentally triggered conversions, which is contrary to usability.

Based on the above, Speicher et al. [7] have pointed out the need for *usability as a target metric* in split tests. This would enable a trade-off between traditional methods and split testing. That is, a usability-based approach to split tests would provide insights into user behavior while leveraging the efficiency and affordability of split testing. As a solution, Speicher et al. [7] propose a novel concept called *Usability-based Split Testing* that must meet three requirements: **(R1)** It is more effective in measuring usability than conversions; **(R2)** Efforts for users and developers are kept to a minimum; and **(R3)** It delivers precise/easy-to-understand usability metrics that enable quantitative comparison of interfaces.

[1] `http://visualwebsiteoptimizer.com/` (2014-03-07).
[2] `http://www.nngroup.com/articles/putting-ab-testing-in-its-place/` (2014-03-16)

In this demo paper, we present *WaPPU* ("*Wa*s that *P*age *P*leasant to *U*se?"), a tool that has been developed for realizing the above concept. Our tool enables developers to perform usability-based A/B tests and carry out metric-based comparisons of web interfaces. For this, WaPPU tracks user interactions $\boldsymbol{I}$ and, based on existing machine learning classifiers, trains models M_i for predicting aspects of usability U_i from these: $M_i(\boldsymbol{I}) = U_i$. The tool offers a dedicated service for creating and monitoring A/B testing projects in real-time with minimal effort.

2 Related Work

WaPPU is related to a variety of existing research and tools concerned with usability evaluation. The following gives a representative overview.

AttrakDiff[3] is an existing A/B testing tool for optimizing e-commerce products. It measures two dimensions of user experience based on a specific instrument. Despite superficial similarities, the tool is conceptually different from WaPPU. Particularly, it does not leverage user interactions for predicting usability metrics based on models.

W3Touch [5] and *m-pathy*[4] are two tools that leverage user interactions for evaluating web interfaces. The first engages interactions to automatically adapt interfaces for touch devices. The latter collects interactions for manual interpretation by an evaluator. In analogy, tools like *WUP* [1] allow to track interactions on a website and compare them to predefined, optimal logs. Yet, all of these do not focus on usability as a quantitative target metric.

Nebeling et al. [4] derive quantitative metrics from static layout properties of a web interface, such as the amount of too small text. The specific aim of their approach is to support the adaptation of interfaces for large-screen contexts. Although they take a step into the direction of usability metrics, they do not leverage dynamic interactions. Contrary to WaPPU, their concept is based on well-grounded *a priori* assumptions rather than learning indicators of good/bad usability from actual user behavior.

3 WaPPU

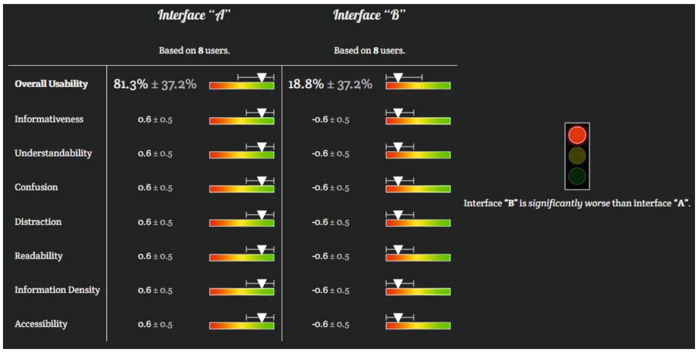
Our tool is realized as a service with a central repository, which enables developers to create and monitor A/B testing projects. First, an A/B testing project is given a name and password for access to real-time analyses and visualizations. Subsequently, the developer chooses which interaction features shall be tracked for which components (defined by jQuery selectors) of their interface.

Listing 1.1. Exemplary WaPPU configuration.

```
WaPPU.start({
    projectId: 42, interfaceVersion: 'A', provideQuestionnaire: true },{
    '#nav': ['hovers', 'hoverTime'],
    '#content': ['cursorSpeed', 'cursorTrail'] });
```

[3] `http://attrakdiff.de/` (2014-03-07).

[4] `http://www.m-pathy.com/cms/` (2014-03-07).



	Interface "A" Based on 8 users.	*Interface "B"* Based on 8 users.
Overall Usability	81.3% ± 37.2%	18.8% ± 37.2%
Informativeness	0.6 ± 0.5	-0.6 ± 0.5
Understandability	0.6 ± 0.5	-0.6 ± 0.5
Confusion	0.6 ± 0.5	-0.6 ± 0.5
Distraction	0.6 ± 0.5	-0.6 ± 0.5
Readability	0.6 ± 0.5	-0.6 ± 0.5
Information Density	0.6 ± 0.5	-0.6 ± 0.5
Accessibility	0.6 ± 0.5	-0.6 ± 0.5

Interface "B" is *significantly worse* than interface "A".

Fig. 1. The WaPPU dashboard

It is possible to choose from a range of 27 predefined interaction features including *clicks*, *length of the cursor trail*, *amount of scrolling* etc. The developer is then provided with two automatically generated JavaScript snippets to be added to the two interfaces-under-test of the A/B testing project (Listing 1.1). The above snippet defines an interface-under-test associated with the split testing project with project ID 42 and specifies that a usability questionnaire is shown to users of the interface. The *amount of hovers* and *hover time* are tracked for the component `#nav`. Moreover, the *length of the cursor trail* as well as the *cursor speed* are tracked for the component `#content`.

By default, the first interface is configured to show a dedicated usability questionnaire *(R1)* with seven items based on [8] (Fig. 1) while the second interface does not. From users' answers to this questionnaire and client-side interactions, WaPPU incrementally learns models (e.g., *Naïve Bayes*[5]) on a per-item and per-context[6] basis for the A/B testing project. These are stored in the central repository. The models are then used to predict the usability of the second interface based on user interactions only *(R1)*. In this way, the amount of questionnaires shown to users is kept to a minimum *(R2)*. Yet, WaPPU can also be configured to show the questionnaire on both (for remote asynchronous user testing) or none of the interfaces-under-test (for use with existing models/heuristics). As soon as the configuration snippets have been integrated into the interfaces *(R2)*, analyses are available via the WaPPU dashboard (Fig. 1). On the dashboard, developers are presented with users' evaluations of seven usability items for the two interfaces. Depending on whether an interface shows a questionnaire, these evaluations are either based on users' answers or on predictions by the learned models for each item. The items' value range lies between -1 and $+1$. The dashboard provides the average rating across all users of the interface along with standard deviations. From the seven individual ratings, WaPPU computes an

[5] `http://weka.sourceforge.net/doc.dev/weka/classifiers/bayes/NaiveBayesUpdateable.html` (2014-03-16).

[6] Context is determined based on screen size and whether an ad blocker is activated. These factors influence the appearance of an interface and thus also users' interactions.

overall usability rating between 0% and 100% *(R3)*. Moreover, the tool automatically performs a *Mann–Whitney U test* to determine whether there is a statistically significant difference between the interfaces w.r.t. overall usability. The result of this test is indicated by a traffic light *(R3)*.

4 Example Study

WaPPU was used to evaluate two versions of a real-world search engine results page in a user study with 81 participants [7]. The old version of the interface had been redesigned by three usability experts. Our tool was able to detect the predicted difference in usability with statistical significance for the largest and most representative user group (screen=*HD*, new user). We were able to train reasonably good models with F-measures between 0.518 (item *distraction*) and 0.296 (item *readability*). Finally, heuristics for search engine results pages were derived based on these models. These state that, e.g., less *confusion* is indicated by a lower scrolling distance from top (Pearson's r=-0.44); and better *readability* is indicated by a lower length of text selections (r=-0.39).

5 Conclusion

We have presented *WaPPU*, a tool for realizing the novel concept of Usability-based Split Testing [7]. Our tool enables developers to set-up A/B testing projects with minimal effort and instantly provides them with quantitative usability metrics. These metrics can be derived from user interactions based on previously learned models. A user study with a real-world interface underpins the feasibility of WaPPU. Potential future work includes integration with existing approaches such as *WebComposition* [2] for automatic optimization of widget-based interfaces.

Acknowledgments. This work has been supported by the ESF and the Free State of Saxony.

References

1. Carta, T., Paternò, F., de Santana, V.F.: Web Usability Probe: A Tool for Supporting Remote Usability Evaluation of Web Sites. In: Campos, P., Graham, N., Jorge, J., Nunes, N., Palanque, P., Winckler, M. (eds.) INTERACT 2011, Part IV. LNCS, vol. 6949, pp. 349–357. Springer, Heidelberg (2011)
2. Gaedke, M., Gräf, G.: Development and Evolution of Web-Applications using the WebComposition Process Model. In: WWW9-WebE Workshop (2000)
3. Insfran, E., Fernandez, A.: A systematic review of usability evaluation in web development. In: Hartmann, S., Zhou, X., Kirchberg, M. (eds.) WISE 2008. LNCS, vol. 5176, pp. 81–91. Springer, Heidelberg (2008)
4. Nebeling, M., Matulic, F., Norrie, M.C.: Metrics for the Evaluation of News Site Content Layout in Large-Screen Contexts. In: Proc. CHI (2011)

5. Nebeling, M., Speicher, M., Norrie, M.C.: W3Touch: Metrics-based Web Page Adaptation for Touch. In: Proc. CHI (2013)
6. Nielsen, J., Molich, R.: Heuristic Evaluation of User Interfaces. In: Proc. CHI (1990)
7. Speicher, M., Both, A., Gaedke, M.: Ensuring web interface quality through usability-based split testing. In: Casteleyn, S., Rossi, G., Winckler, M. (eds.) ICWE 2014. LNCS, vol. 8541, pp. 89–106. Springer, Heidelberg (2014)
8. Speicher, M., Both, A., Gaedke, M.: Towards Metric-based Usability Evaluation of Online Web Interfaces. In: Mensch & Computer Workshopband (2013)

Webification of Software Development: User Feedback for Developer's Modeling

Eduard Kuric and Mária Bieliková

Faculty of Informatics and Information Technologies,
Slovak University of Technology, Ilkovičova 2, 842 16 Bratislava 4, Slovakia
name.surname@stuba.sk

Abstract. In this paper we present an approach to leveraging experience from rapidly evolving field of information processing on the Web for software development. We consider a web of software artifacts (components) as an information space. Supporting any task in such environment of interconnected artifacts depends on our knowledge on user preferences and his characteristics. We envision the concept of collaborative software development to improve software quality and development efficiency by using both implicit and explicit user (software developer) feedback. It opens a space for using approaches originally devised for the Web. The core of our approach is based on our developed platform for independent code monitoring where we create a dataset of developers' implicit and explicit feedback based on monitoring developers' behavior. Employing this platform we acquire, generate and process descriptive metadata that indirectly refer source code artifacts, project documentations and developers activities via document models and user models. As an example of our concept we present an approach for estimation of student's expertise in a programming course.

Keywords: webification of software development, implicit/explicit feedback, interaction information, user modeling, monitoring user behavior.

1 Developer's Feedback in a Web of Software Artifacts

Developers often use a web of software artifacts as a giant repository of source code, which can be utilized for solving their software development tasks. Supporting any task in such environment of interconnected artifacts depends on our knowledge on user preferences and his characteristics. Relevance user feedback is typically used for user profiling during long/short-term modeling of user's interests and preferences on the Web. Relevance feedback techniques have been used to retrieve, filter and recommend a variety of items [4]. Our aim is to support software development by using both implicit and explicit user (software developer) feedback, which creates rich interconnections between software artifacts. This includes not only the shift towards the use of web-based resources in software processes, but also and more importantly it opens a space for using approaches originally devised for the Web (as a network of interconnected content) to support the software development process.

Our work is a part of a research project called PerConIK[1] (Personalized Conveying of Information and Knowledge). We cooperate with a medium size software company. We focus on support of applications development by viewing a software system as a web of information artifacts. Our aim is devising the right metrics to evaluate software artifacts and to identify particular problems and recommending corrective actions. The core of our approach is based on our developed platform for independent code monitoring [1]. We developed several agents that collect and process documentations, source code repositories, developers' activities, etc. We create within the project a dataset of developers' implicit and explicit feedback based on monitoring behavior of developers for the purpose of estimating developers' expertise. For example, we exploit implicit feedback such as searching relevant information on the Web and searching in source code during performing tasks, writing and correcting source code in development environment, and explicit feedback such as peer review (review feedback attached to source code).

To the present, we have focused mainly on modeling developer's expertise in software house environment. It is based on investigation of software artifacts which the developer creates and the way how the artifacts were created. In other words, we take into account the developer's source code contributions, their complexity and how the contributions were created to a software artifact (e.g. copy/paste actions from external resources, such as a web browser); the developer's know-how persistence about a software artifact; and technological know-how - the level of how the developer knows the used libraries, i.e., broadly/effectively. All on daily basis of software development.

In a software company estimation of developers' expertise allows, for example, on the one hand, managers and team leaders to look for specialists with desired abilities, form working teams or compare candidates for certain positions, on the other hand, developers can locate an expert in a particular library or a part of a software system (someone who knows a component or an application interface) [2,5]. It can be also used to support so-called "search-driven development". When a developer reuses a software artifact from an external source he has to trust the work of an external developer who is unknown to him. If a target developer would easily see that a developer with a good level of expertise has participated in writing the software artifact, then the target developer will be more likely to think about reusing.

On the contrary of a software company, where software is created by professionals, in academic environment, students learn how to design and develop software. Moreover, a student produces significantly less data (implicit/explicit feedback). Our goal is to provide a tool that allows a teacher to evaluate student's knowledge and skills (expertise) based on monitoring student's behavior during developing tasks and adaptation of instruments developed for the general approach. As an example of our approach we present an approach for estimation of student's expertise in a programming course. It allows, for example, teacher to adapt and modify his teaching practices.

[1] PerConIK: `http://perconik.fiit.stuba.sk/`

2 Estimation of Student's Programming Expertise

In our approach modeling student's expertise is based on estimation of a degree of student's expertise of a concept in comparing with other investigated students. In other words, if students solve a task focused on acquiring skills of particular concept, e.g., *priority queue*, then by analyzing their resultant source code, interaction data and by using appropriate software metrics, we can estimate levels of students' expertise of the concept. By using the particular students' expertise estimations of concepts we are able to estimate a degree of student's expertise for the whole course and compare the estimations among the investigated students based on the same evaluation criteria.

We experimented with data gathered during bachelor course on *Data structures and algorithms.* During seminars the students solve programming tasks. Each week is focused on training and acquiring skills of a concept such as *stack, binary tree, hash table, etc.* The students solve the tasks in a learning system *Peoplia*[2]. Students can select to solve a simpler or more complex task focused on acquiring skills of a concept. Students get points for their successful solutions. In autumn semester 2013/14, 251 students enrolled in the course.

When a student submits a solution of a task to *Peoplia*, its correctness and efficiency (time complexity) is evaluated. The solution is accepted if it is correct and efficiency tests are successful. The student has unlimited number of submission attempts and the solutions are checked by a plagiarism detection system. Estimation of a degree of student's expertise of a concept c based on a student's correct solution l for a programming task t is calculated as follows:

$$Exp_c(s,t,l) = CX(t) * EF(s,l) * \frac{1}{log_2(1 + CT(s,t))}, \tag{1}$$

where $CX(t)$ is complexity of the task t estimated based on a combination of Logical Source Lines of Code (SLOC-L) and McCabe VG complexity metrics. For calculation of SLOC-L we have adopted the definition from the CodeCount[3]. $EF(s,l)$ returns 1.5 if the student's s submitted solution l is effective, otherwise 1. A solution is effective if its execution time is less than or is equal to a median value of all execution times of submitted correct solutions for t (it is based on preliminary experiments). $CT(s,t)$ is a number of submitted solutions by the student s for the task t (the last solution was accepted by Peoplia). The estimation of a degree of student's expertise of the course is calculated as $\sum_i Exp_{c_i}(s,t_i,l_i)$.

We estimated expertise for all students and compared our results to results achieved on exam. 78 out of 251 students were not allowed to take the final exam because they did not achieve the qualification criteria. Both values (estimated expertise and points of the final exam) were normalized into the interval $[0,100]$. To each student a pair (X,Y) is assigned, where $X \in [0,100]$ is a number of points of the final exam and $Y \in [0,100]$ is the estimated student's expertise. Subsequently, the values in each pair were mapped as follows: A – $[92,100]$, B – $[83,91]$, C – $[74,82]$, D – $[65,73]$, E – $[56,64]$, and FX – $[0,55]$.

[2] Peoplia: `http://www.peoplia.org/`
[3] USC CodeCount: `http://sunset.usc.edu/research/CODECOUNT/`

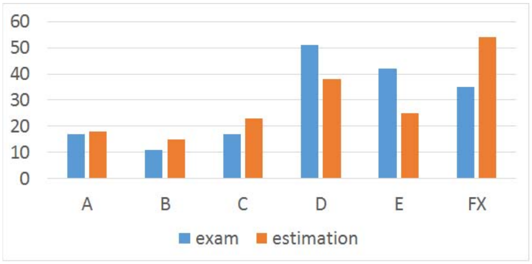


Fig. 1. Comparison of how automatic estimation of students' expertise correlates with exam results

The result of our experiment is illustrated in Figure 1. The number of concordant pairs equals 143 and the number of discordant pairs equals 30. The total number of pairs equals 173. We calculated precision as 0.83.

The idea of "webification" of software development which is based on viewing software repositories as webs is not new. Already Knuth in 1984 presented the idea that "a program is best thought of as a web"[3]. The novel aspect lies in considering not only software artifacts but also users (developers) together with their explicit and implicit feedback, which brings a new view on software and software process metrics. It helps developers be more efficient and can enrich them with the experience and knowledge of their colleagues while managers or senior developers can get advantage of improved planning and decision support via aggregation of statistical data for individual developers.

Acknowledgement. This work was partially supported by the Scientific Grant Agency of the Slovak Republic, grant No. VG1/0675/11 and it is the partial result of the Research & Development Operational Programme for the project PerConIK, ITMS 26240220039, co-funded by the ERDF.

References

1. Bieliková, M., Polášek, I., Barla, M., Kuric, E., Rástočný, K., Tvarožek, J., Lacko, P.: Platform independent software development monitoring: Design of an architecture. In: Geffert, V., Preneel, B., Rovan, B., Štuller, J., Tjoa, A.M. (eds.) SOFSEM 2014. LNCS, vol. 8327, pp. 126–137. Springer, Heidelberg (2014)
2. Fritz, T., Ou, J., Murphy, G.C., Murphy-Hill, E.: A degree-of-knowledge model to capture source code familiarity. In: Proc. of the 32nd Int. Conf. on Softw. Eng., vol. 1, pp. 385–394. ACM, USA (2010)
3. Knuth, D.E.: Literate programming. Comput. J. 27(2), 97–111 (1984)
4. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, USA (2008)
5. Minto, S., Murphy, G.C.: Recommending emergent teams. In: Proc. of the 4th Int. Workshop on Mining Softw. Repositories. IEEE Computer Society, USA (2007)

Comparing Methods of Trend Assessment

Radek Malinský and Ivan Jelínek

Department of Computer Science and Engineering, Faculty of Electrical Engineering
Czech Technical University in Prague,
Karlovo náměstí 13, 121 35 Prague, Czech republic
{malinrad,jelinek}@fel.cvut.cz
http://webing.felk.cvut.cz

Abstract. This paper deals with a comparison of selected webometric methods for the evaluation of Internet trends. Each of the selected methods uses a different methodology to the trend assessment: frequency, polarity, source quality. It can be assumed that a combination of individual methods can provide much more accurate results with respect to the desired area of interest. This will lead to improve the quality of search engines on the principle of webometrics and thereby the reduction of irrelevant web search results. The introductory part of the paper explains a concept and basic functional background for all selected webometric methods.

Keywords: Webometrics, Web Mention Analysis, Sentiment Analysis, Social Network Analysis, Trend Assessment.

1 Introduction

With the growing popularity of social networking and blogging, there have been arising a large number of comments on various topics from many different types of users on the web. Some of these comments may be totally unimportant to the other Internet users. On the contrary, other comments might be very important, and do not only for an ordinary user, but also for some commercial companies that want to know a public opinion on price, quality and other factors of their products.

However, web content diversity, variety of technologies and website structure differences, all of these make the web a network of heterogeneous data, where things are difficult to find for common internet users.

Web search engines are the easiest way to find specific information in such diversified network for ordinary users. The search engines are based on complex algorithms that allow search structured but also unstructured data sets and return the most relevant results in a correlation to user-entered query. Webometric methods are often used as supportive assessment methods for search engines algorithms; Web Mention Web Analysis, Sentiment Analysis and Social Network Analysis are among the most widely used webometric methods.

Each of the selected methods uses a different methodology to the trend assessment: frequency, polarity, source quality. It can be assumed that a combination of individual methods can provide much more accurate results with respect to the desired area of interest. This will lead to improve the quality of search engines on the principle of webometrics and thereby the reduction of irrelevant web search results.

2 Selected Methods of Trend Assessment

Web Mention Analysis, Sentiment Analysis and Social Network Analysis are among frequently used methods for searching and evaluating of web pages [7]. The selected methods were primarily chosen for their diversity and applicability in various areas of web and social engineering; blogs and social networks are an ideal data source for social science research because it contains vast amount of information from many different users.

Web Mention Analysis [7] is used for the evaluation of the “web impact” of documents or ideas by counting how often they are mentioned online. This idea essentially originates due to a study of an academic research. The researches wanted to know the place and the context which their works occurred in. This approach is applied in a commercial search service Google Scholar [1], which covers not only academic works but also journal articles, patents, etc. Another example of the use of Web Mention Analysis is an identification of how often and in which countries is some product (e.g. camera, book) mentioned online [2].

Sentiment Analysis or Opinion Mining [3], [4] enables us to automatically detect opinions from structured but also unstructured data. The research in this field originated from the demand of commercial companies, who wanted to know public opinion on price, quality and other features of their products. The classification of sentences or whole documents is very often based on the identification of sentiments of individual words or phrases [5]. Several approaches for the purpose have been explored and basically divided into three categories [8]: full-text machine learning, lexicon-based methods and linguistic analysis.

Social Network Analysis is the mapping and measuring of relationships on the web [2], [6]. A centrality is a part of the social network analysis, which is very often used in the web link analysis [6]. The centrality is a single node feature, which explains the node position in a network. It is for example used to determine the most active collaborator in the collaboration scientific networks [2]. There are several methods to measure the centrality of the nodes in a network [6].

3 Methodology of Study

The methods selected for the evaluation of the trends were compared over the data from film industry. Blog posts published in 2012-2013 served as the data source for this research. The five best rated movies which premiered in 2012 were chosen as trends for the assessment. All movies were selected according to IMDb[1]

[1] IMDb (Internet Movie Database) - an online database of information related to films, `http://www.imdb.com`

ranking, which is based on the site visitors rating. The rating is performed by selecting a numeric value from 1 to 10; with 10 being the best.

Because it is very difficult to find a correlation among the methods, the output of each assessment is represented as a list of movies rated from best (1) to worst (5). Table 1 shows the evaluation of trends for each compared method. The names of movies were for all comparing methods used as exact phrase searched with quotes on both sides, e. g. "The Avengers".

Table 1. Comparing Methods of Trend Assessment

Movie	IMDb	SA	WMA	SNA	SNA+SA	Rank
Django Unchained	2	3	5	4	3	4 (12)
Life of Pi	4	5	4	5	5	5 (14)
The Avengers	3	2	3	2	4	2 (7)
The Dark Knight Rises	1	1	1	1	1	1 (3)
The Hobbit: An Unexpected Journey	5	4	2	3	2	3 (9)

For Sentiment Analysis (SA), the surroundings of each searched expression had been recognized and a list of sentences for the trend had been created. All sentences were processed using Lexicon-Based method with SentiWordNet as a lexicon of words. Web Mention Analysis (WMA) is based on counting how often searched words were mentioned in the corpus of blog posts. For Social Network Analysis (SNA), the Degree Centrality was used to determine the most read blog posts and thereby to identify the trend assessment. The combination of methods (SNA+SA) represents the evaluation of trends by using Sentiment Analysis for only the most important blogs that were selected in the previous step through Social Network Analysis.

Values in the column "Rank" were determined by the sum of (SA)+(WMA)+(SNA); result of the sum is given in parentheses. Trend assessment is represented by the first number; lower sum means better ranking. The combination of methods (SNA+SA) had not been included into the sum because its output does not reflect all the data, but only the selected most important blogs are evaluated. The final ranking in the "Rank" column in comparison with ranking in the "IMDb" column, it represents the rating difference between "common users" and "film fans from IMDb".

The result shows that movie *The Dark Knight Rises* was rated as the best by all methods; it is also correlated with ranking from IMDb. On the contrary, *Django Unchained*, very well ranked on IMDb, it did not gain too much popularity on blogs (Rank is 4). This may be caused primarily by the value of WMA, which shows that this movie was at least mentioned on the web in comparison with other movies. Another important influence of the overall assessment by WMA is observed on the *The Hobbit: An Unexpected Journey.* This movie was the worst rated of the selected set of movies from IMDb. This negative evaluation is also reflected by SA, which shows the high number of negative words. However, the low value of WMA proves that the movie was high interested.

4 Conclusion

We have selected three webometric methods, which are often used as supportive search engines assessment algorithms. Each of the selected methods was used to analyze five trends (movie titles) over a set of blog posts published in 2012-2013. The output of the analysis is by popularity ordered ranking of trends (movies).

The output of each method represents a different view on the evaluation of trends: Web Mention Analysis - emphasizes the frequency of blog posts that mention the trend; Sentiment Analysis - defines the output based on the positive / negative feedback from bloggers; Social Network Analysis – defines the output by quality of blogs that mention the trend. The combination of individual methods can provide much more accurate results with respect to the desired area of interest. In our case the ranking defined by the all three methods in comparison with ranking from IMDb represents the rating difference between "common users" and "film fans from IMDb".

The subject of future work is especially in the finding a correlation among the methods. This means to define criteria for quality assessment of found information, and "distance" among each trend. On this basis, rules for evaluation of semantic content in relation to user's queries can be designed.

Acknowledgments. This research has been supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS12/149/OHK3/2T/13.

References

1. Gehanno, J. F., Rollin, L., Darmoni, S.: Is the coverage of Google Scholar enough to be used alone for systematic reviews. BMC medical informatics and decision making. Vol 13, No. 1 (2013)
2. Han, S. K., Shin, D., Jung, J. Y., Park, J.: Exploring the relationship between keywords and feed elements in blog post search. World Wide Web. 12:381–398 (2009)
3. Jagtap, V. S., Pawar, K.: Analysis of different approaches to Sentence-Level Sentiment Classification. In International Journal of Scientific Engineering and Technology. Vol. 2, No. 3, 164-170 (2013)
4. Liu, B.: Sentiment analysis and opinion mining. In Synthesis Lectures on Human Language Technologies. Vol. 5, No. 1, 1-167 (2012)
5. Montejo-Ráez, A., Martínez-Cámara, E., Martín-Valdivia, M. T., Urena-López, L. A.: Ranked WordNet Graph for Sentiment Polarity Classification in Twitter. In Computer Speech & Language. Vol. 41, No. 11, 373-381 (2013)
6. Pak, A., Paroubek, P.: Twitter as a corpus for sentiment analysis and opinion mining. Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10), Valletta, Malta (2010)
7. Thelwall, M.: Introduction to webometrics: Quantitative web research for the social sciences. San Rafael, CA: Morgan & Claypool (2009)
8. Thelwall, M., Buckley, K., Paltoglou, G.: Sentiment in twitter events. Journal of the American Society for Information Science and Technology. 62:406–418 (2011)

Exploiting Different Bioinformatics Resources for Enhancing Content Recommendations

Abdullah Almuhaimeed* and Maria Fasli

University of Essex
{ansalm,mfasli}@essex.ac.uk

Abstract. To assist the user in his/her quest for information it may be possible to draw and combine information from multiple resources in order to provide more accurate answers/recommendations. Resources can be structured (such as ontologies and taxonomies) or unstructured (corpora). The purpose of this work is to explore how better recommendations can be provided to users by mining and exploiting semantic relations, hidden associations, and overlapping information between various concepts in multiple bioinformatics resources such as ontologies, websites, and corpora. The work also utilizes users' interests to enhance the provided recommendations. A number of techniques will be explored and developed, including ontology mapping, reasoning with multiple resources, and constructing adaptive user profiles.

Keywords: Semantic Techniques, Recommendations & Bioinformatics.

1 Introduction

Given the recent advances in the field of Bioinformatics, a lot more information has become available online. However, searching for such information may not necessarily be easy as resources remain unconnected and current search engine and recommendation systems are not able to combine information that may exist in different resources in order to better understand the user request, enrich it and then use it in order to extract more accurate information to satisfy the users' needs. For instance, Middleton et al. [5] developed a recommender approach which provides recommendations on online academic papers. It uses a single source (i.e. ontology) to enrich a user profile and draws recommendations based on this enrichment. This approach does not take into account the availability of multiple sources of information (ontologies, taxonomies, etc.) to enrich the user profile, and as a result this may decrease the accuracy of the provided recommendations. The motivation of this work is to bridge this gap by using multiple sources to provide the user with more accurate and rich recommendations. We also aim to develop techniques that infer semantic relations and hidden associations from different bioinformatics resources which we subsequently exploit to enhance the precision of the provided recommendations, while we also make use of user profiles to further tailor the recommendations provided.

* PhD student, School of Computer Science & Electronic Engineering.

2 Background

Ding et al. [1] purports that ontologies are a fundamental concept in the Semantic Web, used to represent researcher perspectives about a domain in a conceptualised manner. A number of formal languages have been developed to represent ontologies such as Resource Description Framework (RDF), Resource Description Framework Schema (RDFs) [6], and Web Ontology Language (OWL) [3]. Reasoning with multiple resources is an increasing need due to the large amount of resources contained on the Web. There are several challenges that need to be overcome in performing reasoning through different resources. These resources may have different structures and may not necessarily be compatible, which may lead to inconsistencies during the reasoning process [2]. In addition to exploiting semantic relations between bioinformatics resources, user profiles represent an important source of information providing recommendations to users. User profiles can be constructed based on different methods. The data used to construct user profiles can be classified into two types: static and dynamic data. Static which does not change very frequently, such as names. Dynamic information represents user preferences that can be collected explicitly or implicitly [4].

3 The Recommender Approach

This research contributes to developing semantic-based methods for identifying relations and hidden associations extracted from bioinformatics resources (e.g. ontologies such as Protein Ontology (PO)[1], Gene Ontology (GO)[2], Open Directory Project (ODP)[3] and Bioinformatics Links Directory (BLD)[4] and corpora such as Wikipedia). By studying such resources, we have concluded that there is implicit information that can be extracted through semantic analysis and our central hypothesis is that this can be used in providing better recommendations to users. In addition, we want to tailor-make the recommendations to user needs based on their profiles which will be automatically constructed by collecting user preferences and interests implicitly. We aim to demonstrate our methods through providing recommendations to bioinformaticians on the most relevant content (i.e. articles) from the bioinformatics corpus BMC[5].

Our work consists of two branches: (i) developing methods for extracting semantic information from multiple resources (such as ontologies, taxonomies, Wikipedia) and reasoning with this information to obtain new relations; (ii) constructing an ontological user profile based on information extracted implicitly from the user surfed sessions as well as interaction with the system. The information from (i) and (ii) is then combined to enrich the user query and provide more accurate recommendations to users.

[1] `http://pir.georgetown.edu/pro/`
[2] `http://geneontology.org/`
[3] `http://www.dmoz.org/`
[4] `http://bioinformatics.ca/links_directory/`
[5] `http://code.google.com/p/bmc-bioinformatics-processed-corpus/`

We first aim to develop a reasoning method to exploit overlapping information between different bioinformatics resources such PO[1], GO[2], etc., and to extract semantic relations and hidden associations between different classes. This method uses SPARQL[6] queries to extract information, and provides them to the reasoner which combines them with semantic rules to infer new relations that may exist among resources. As a result, a semantic network is created which represents the extracted semantic relations and hidden associations from the intersection between different resources. This includes new identified relations, not found in the original resources. Users' profiles will then be boosted by adding the relevant information from this network.

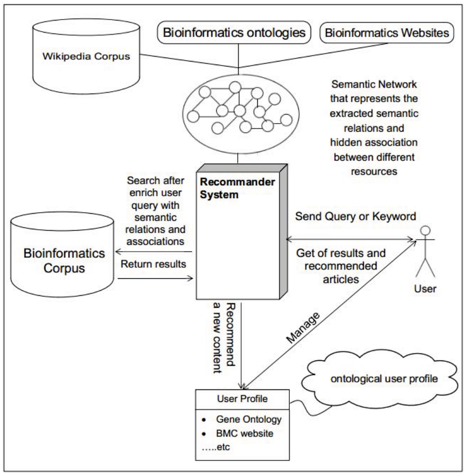

Wikipedia Corpus
Bioinformatics ontologies
Bioinformatics Websites
Semantic Network that represents the extracted semantic relations and hidden association between different resources
Recommander System
Search after enrich user query with semantic relations and associations
Return results
Bioinformatics Corpus
Send Query or Keyword
Get of results and recommended articles
User
Recommend a new content
Manage
ontological user profile
User Profile
• Gene Ontology
• BMC website
.....etc

Fig. 1. Recommender System Structure

We suppose that such enrichment contributes to enhancing the precision and accuracy of the returned results and recommendations. There are several challenges that need to be overcome in this branch of work in that the resources' structure may contain inconsistencies, resources may get updated or further developed and hence the system needs to take this into account, and exploiting semantic relations to enhance recommendations may not be straightforward. The contribution of our approach will be in its ability to handle different resources with various structures such as ontologies and corpora and employing extracted semantic relations, such as siblings relations, between multiple resources to enhance the accuracy of the articles' recommendations.

Among the requirements to reach our goal of providing a user with recommendations that are drawn from multiple resources is constructing an adaptive ontological profile based on a bioinformatics ontology (i.e. ODP[3] bioinformatics branch). This profile will be equipped with mechanisms to perform the main tasks (i.e. add, update, and delete). All of the aforementioned tasks will be handled automatically. An adaptive user profile needs to be created and examined in a manner that reflects the user's interests. This profile should be able to accommodate the frequent changes of the user preferences, the enrichment with valuable information that is gained from the semantic network and providing fully automated solutions. This approach will be fully automated and tailor recommendations to each user individually based on his/her preferred topics.

[6] http://www.w3.org/TR/rdf-sparql-query/query

4 Bioinformatics Recommender Pipeline

For assessing our recommendation approach, a system pipeline (Figure 1) was put together as follows: a user is interested in identifying articles of interest and will enter a bioinformatics concept (query) to be searched in the BMC[5] corpus. We use the Lucene[7] search engine to index and retrieve data from the corpus. A method has been created to automatically collect user's interests from surfed sessions and we subsequently calculate their similarity with the ODP[3] ontology to construct ontological user profiles. The semantic network will enrich the user's query with valuable information that is acquired from multiple resources. If the same class has been found in more than one resource, our approach can exploit any associated extra information but avoids duplication and repetition. The results (i.e. articles) are re-ranked and returned based on the highest degree of similarity to the user's preferences and the enrichment of the query that is provided by the semantic network. In this pipeline, we target the recommender system, semantic network and user profile. In addition, each user is presented with a hyperlink which contains some recommendations related to the entered query as well as the top five interests stored in the user profile.

5 Conclusion

This research aims to provide a set of new methods to improve recommendations. This includes a personalised recommender service, a mechanism for reasoning through multiple resources and extracting semantic relations and hidden associations. This service should include modelling and learning automatic adaptive user profiles, a method for representing semantic relations and hidden associations, and a method for filtering user interests. Finally, we have constructed the main blocks of our recommender approach to assess to what extent our assumption assists in enhancing the precision of the provided recommendations. We are currently preparing to run an experiment with the aid of bioinformaticians.

References

1. Ding, L., Kolari, P., Ding, Z., Avancha, S.: Using ontologies in the semantic web: A survey. In: Ontologies, pp. 79–113. Springer (2007)
2. Huang, Z., Van Harmelen, F.: Using semantic distances for reasoning with inconsistent ontologies. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 178–194. Springer, Heidelberg (2008)
3. McGuinness, D.L., Van Harmelen, F., et al.: OWL web ontology language overview. W3C Recommendation 10(2004-03) (2004)
4. Mezghani, M., Zayani, C.A., Amous, I., Gargouri, F.: A user profile modelling using social annotations: a survey. In: WWW 2012, pp. 969–976 (2012)
5. Middleton, S.E., Shadbolt, N.R., De Roure, D.C.: Ontological user profiling in recommender systems. ACM Transactions on Information Systems 22(1), 54–88 (2004)
6. Miller, E.: An introduction to the resource description framework. Bulletin of the American Society for Information Science and Technology (1998)

[7] `https://lucene.apache.org/core/`

Methodologies for the Development of Crowd and Social-Based Applications*

Andrea Mauri

Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)
Politecnico di Milano. Piazza Leonardo da Vinci, 32. 20133 Milano, Italy
andrea.mauri@polimi.it

Abstract. Even though search systems are very efficient in retrieving world-wide information, they cannot capture some peculiar aspects of user needs, such as subjective opinions, or information that require local or domain specific expertise. In these scenarios the knowledge of an expert or a friend's advice can be more useful than any information retrieved by a search system. This way of exploiting human knowledge for information seeking and computational task is called Crowdsourcing. The main objective of this work is to develop methodologies for the creation of applications based on Crowdsourcing and social interaction. The outcome will be a framework based on model-driven approach that will allow end user to develop their own application with a fraction of the effort required by the traditional approaches. It will guarantee a strong control of the execution of the crowdsourcing task by mean of a declarative specification of objectives and quality measures. A prototype will be developed that will allow the creation and execution of task on various platforms. Validation of the approach will consist of quantitative and qualitative analysis of results and performance of the system upon some sample scenarios, where real users from social networks and crowdsourcing platforms will be involved.

1 Introduction

Crowd-based applications are becoming more and more widespread [4]; their common aspect is that they deal with solving a problem by involving a vast set of performers, who are typically extracted from a wide population (the "crowd"). In a typical crowdsourcing scenario is presente a *requester*, who is the one who want to solve a problem with a crowdsourcing campaign (a set of one or more tasks created to fulfill an objective), a set of *responders*, which are the people who perform some tasks and provide answers, and finally the *system* that organize the tasks and collect the results.

The main objective of the requester is to solve his problem while making the best use of responder's availability and reliability so as to get the best possible result for his campaign. For instance he wants to maximize the quality while minimizing the cost and time.

* This research is developed under the supervision of Professor Marco Brambilla.

Crowdsourcing is applied to different fields and within various communities, including information retrieval, databases [5], artificial intelligence and social sciences.

Moreover different platforms can be used to perform this activity, ranging from the classical crowdsourcing marketplaces (e.g Amazon Mechanical Turk), to question answering systems (e.g Quora[1]) and generic social networks (e.g Facebook).

Large crowds may take part to human computation for a variety of motivations, which include non-monetary ones, such as public recognition, fun, or the genuine wish of contributing their knowledge to a social process.

In this situation different issues arise: each type of scenario is characterized by a peculiar set of needs and requirements, that need to be mapped to the particular platform,that usually is not flexible, as it do not support a high-level, fine-tuned control upon posting and retracting tasks.

For instance if the requester wants to post and control a crowdsourcing task on Amazon Mechanical Turk he has to code the implementation with imperative and low-level programming language or using a framework like Turkit [10]. If he wants to exploit the relations between people, he may want to use a social network as crowdsourcing platform. In this case the requester has to directly use the API provided by the social network.

The objective of this work is to develop methodologies for creating applications that leverage the knowledge of the crowd or social communities. The approach developed should be platform agnostic and allow the requester to create his application without having strong technical knowledge.

Thus the research questions that lead this work are: (1)What are the main features of a crowdsourcing campaign ?(2)How can I abstract all these characteristics in a agnostic metamodel? (3) How can this model be used to facilitate the development of a crowd or social based application maximizing its performance?

2 The Approach

In this work I define a top-down approach to application design that adopts an abstract model of crowdsourcing activities in terms of elementary task operations (such as: labeling, liking, sorting, classifying, grouping) performed upon a data set. Starting from operations, strategies are defined for task splitting, replication, and assignment to performers. I also define the data structures which are needed for controlling the planning, execution, and reactive control of crowd-based applications.

The whole process follow a model driven approach, in which the data structures needed for the execution of the application are automatically generated starting from the abstract description of the crowdsourcing task.

Figure 1 shows an example of such process (taken from [3]). On the top there is the model, while on the bottom there are the instances. The image shows how the model-driven process generates the structure needed for the execution and control. The **metamodel**(1) describes which operation needs to be performed

[1] `www.quora.com`

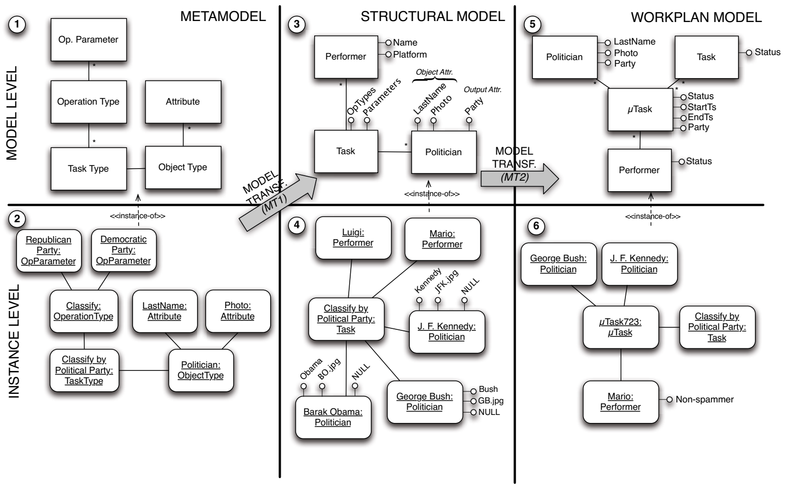


Fig. 1. The model transformations that generate the needed data structure starting from the abstract model

and the schema of the data to be used. The first model transformation generate the **structural model**(3), that contains all the information needed for the execution of the task. Then the second transformation generate the structure needed for the control (the **workplan model**(5)) . This model is composed by the **control mart**, that is analogous to data marts used for data warehousing [6] as its central entity represents the facts, surrounded by three dimension tables, and the **aggregate tables** (Performer, Object and Task), that contain aggregate information.

Control is performed on top of this last model by mean of declarative rules. A rule is composed by an event, condition and action. An event is triggered when a change in the control mart occurs, a condition is a predicate that must be satisfied in order to execute the action, finally, an action is a modification of the control mart and aggregate tables.

Together to the conceptual approach, a concrete prototype was developed[2]. Early results were published in [3].

3 Related Work

Most current approaches rely on an imperative programming models to specify the interaction with crowdsourcing services. For instance Turkit [10] offers a scripting language for programming iterative tasks on Amazon Mechanical Turk. RABJ [8] offers very simple built-in control logic, while complex controls are externalized to client applications written in HTML and Javascript. Another example is Jabberwocky [1], a framework that transparently manages several crowdsourcing platforms and can be procedurally programmed.

[2] http://crowdsearcher.search-computing.org/

Other works propose approaches for human computation which are based on high-level abstractions, sometimes of declarative nature. For example in [13] the authors propose a language that interleaves human-computable functions, standard relational operators and algorithmic computation in a declarative way. Qurk [11] exploits a relational data model, SQL to express queries, and a UDF-like approach to specify human tasks. CrowdDB [5] also adopts a declarative approach by extending SQL both as a language for modeling data and to ask queries; human tasks are modeled as crowd operators in query plan, from which it is possible to semi-automatically derive task execution interfaces. Similarly, the DeCo [14] system allows SQL queries to be executed on a crowd-enriched data-source. Finally CrowdLang [12] supports workflow design and execution of tasks involving human and machine activities, it incorporates explicit abstractions for group decision processes and human computation tasks. None of the these works face the problem of specifying the control associated with the execution of human tasks, leaving its management to opaque optimization strategies. Moreover all these systems and researches either deal with a single specific problem (and they focus on a single deployment platform) or they focus on a single aspect of the crowdsourcing campaign.

4 Conclusions and Future Work

The model and the process described in this paper are the starting point of my PhD work. Future steps will consist in extending the conceptual framework in the following way:

– **Crowdsourcing workflow:** In the current model the crowdsourcing campaign is composed by a single task, while complex problems need to be solved by coordinating different tasks. An example of this kind of problem is the creation and modification of content, where is difficult to evaluate and aggregate the outcome of a given task. Previous works face the problem of designing and executing workflow of tasks focusing on a specific domain [2][7] or develop a programmatic approach based on a single execution platform [9]. I aim to start from the existing model of the task extending the existing concepts in order to support task coordination.
– **Higher level of abstraction:** The requester needs to directly configure the task to be performed and has to program himself the control rules (if not provided by the system). Raising the level of abstraction of the model allows to automatically generate the correct strategies and control rules needed to solve the problem.
– **Optimization problem:** There are several possible task configurations suited to solve a particular problem. In this case it's interesting to study the problem of finding the *optimal* configuration. To achieve this, specific metrics are needed in order to evaluate the effectiveness of a selected approach.

In parallel the development of the prototype will be carried on in order to support the evolution of the conceptual model.

References

[1] Ahmad, S., Battle, A., Malkani, Z., Kamvar, S.: The jabberwocky programming environment for structured social computing. In: UIST 2011, pp. 53–64. ACM (2011)

[2] Bernstein, M.S., Little, G., Miller, R.C., Hartmann, B., Ackerman, M.S., Karger, D.R., Crowell, D., Panovich, K.: Soylent: a word processor with a crowd inside. In: Proceedings of the 23rd Annual ACM Symposium on User Interface Software and Technology, UIST 2010, pp. 313–322. ACM, New York (2010)

[3] Bozzon, A., Brambilla, M., Ceri, S., Mauri, A.: Reactive crowdsourcing. In: 22nd World Wide Web Conf., WWW 2013, pp. 153–164 (2013)

[4] Doan, A., Ramakrishnan, R., Halevy, A.Y.: Crowdsourcing systems on the world-wide web. Commun. ACM 54(4), 86–96 (2011)

[5] Franklin, M.J., Kossmann, D., Kraska, T., Ramesh, S., Xin, R.: Crowddb: answering queries with crowdsourcing. In: ACM SIGMOD 2011, pp. 61–72. ACM (2011)

[6] Inmon, W.H.: Building the Data Warehouse. John Wiley & Sons, Inc., New York (1992)

[7] Kittur, A., Smus, B., Khamkar, S., Kraut, R.E.: Crowdforge: Crowdsourcing complex work. In: Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, UIST 2011, pp. 43–52. ACM, New York (2011)

[8] Kochhar, S., Mazzocchi, S., Paritosh, P.: The anatomy of a large-scale human computation engine. In: HCOMP 2010, pp. 10–17. ACM (2010)

[9] Kulkarni, A., Can, M., Hartmann, B.: Collaboratively crowdsourcing workflows with turkomatic. In: Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, CSCW 2012, pp. 1003–1012. ACM, New York (2012)

[10] Little, G., Chilton, L.B., Goldman, M., Miller, R.C.: Turkit: tools for iterative tasks on mechanical turk. In: HCOMP 2009, pp. 29–30. ACM (2009)

[11] Marcus, A., Wu, E., Madden, S., Miller, R.C.: Crowdsourced databases: Query processing with people. In: CIDR 2011, pp. 211–214 (January 2011), `www.cidrdb.org`

[12] Minder, P., Bernstein, A.: How to translate a book within an hour: towards general purpose programmable human computers with crowdlang. In: WebScience 2012, Evanston, IL, USA, pp. 209–212. ACM (January 2012)

[13] Parameswaran, A.G., Polyzotis, N.: Answering queries using humans, algorithms and databases. In: CIDR 2011, Asilomar, CA, USA, pp. 160–166 (January 2011)

[14] Park, H., Pang, R., Parameswaran, A.G., Garcia-Molina, H., Polyzotis, N., Widom, J.: Deco: A system for declarative crowdsourcing. PVLDB 5(12), 1990–1993 (2012)

Using Semantic Techniques to Improve Service Composition by End Users

Giuseppe Desolda

Dipartimento di Informatica, Università degli Studi di Bari Aldo Moro
via Orabona, 4, 70125 - Bari, Italy
giuseppe.desolda@uniba.it

Abstract. My PhD research focuses on supporting non-technical end users to flexibly integrate, into personal interactive workspaces, heterogeneous services available in the Web, in order to satisfy their own personal needs in various contexts of daily life. End users should be enabled to shape their application at use time, and also supported on identifying data and services of interests. The latter requirement addresses an issue about improving the quality of the retrieved services with respect to the end user's goal. As described in this paper, a direction that I am exploring refers to the use of linked open data as a new source of data to be exploited, in order to better fulfill the end user's desires.

Keywords: Composition Paradigms, End-User Development, Linked Data.

1 Introduction

Nowadays we are facing the increasing amount of data and services available on Internet. This situation, together with the opportunities offered by Web 2.0, stimulates researchers to investigate new ways for effectively allowing laypeople, i.e., end users without expertise in programming (often called end users in the rest of this paper), to access, manipulate, and combine different kinds of resources in order to generate personalized contents and applications. Mashups have been very much investigated in the last years; they are applications assembled by end users through the integration of heterogeneous resources (APIs, databases, spreadsheets, HTML pages, etc.), in order to solve situational needs. Unfortunately, the proposed mashup platforms are not very end-user oriented [1]. Some mashup tools provide graphical user interfaces for combining services, but the adopted visual composition languages are not suitable for end users, who have difficulties in understanding the integration logic (e.g. data flow, parameter coupling) [2]. Moreover, platforms are usually general purpose and not adequate to the needs of specific application domains and specific end users.

The research I'm interested for my PhD aims at empowering end users to create personal interactive workspaces by combining End User Development (EUD) principles with mashup methodologies and techniques. In this way, services available on the Web can be composed to create new contents and applications by using intuitive and easy-to-use composition mechanisms and platforms. The first couple of years of my research, I investigated recent proposals of mashup compositions and identified models and techniques derived from the lessons learned on End-User Development; they were used in developing a novel methodology and the prototype of a platform for service and data composition to satisfy end users' needs [3]. A key point is that the platform has to offer mechanisms to be customized to a specific domain in which it has to be used, so that, capitalizing on the knowledge of the people working in that domain, it can offer a composition process that is adequate to such people. Thus, the developed platform is based on a meta-design approach and a novel stratification into different design layers. Meta-design permits the involvement of different stakeholders in the design: the first phase (the meta-design phase) is performed by professional developers and consists of designing software environments that allow some stakeholders (including domain experts) to create templates, basic elements, and software environments appropriate for end users in the specific application domain, making possible the domain customization; in the second phase, using such environments, end users are able to compose their Personal Information Space (PIS), in which they integrate and manipulate services of interest. Thus, as explained in details in [3], there are different design layers: at the top layer, professional developers create software environments, services descriptors and visual templates for other stakeholders by using an integrated development environment; at the middle layer, domain experts and professional developers perform participatory design to customize the platform by selecting visual templates and registering and composing services; at the bottom layer, end users can create, use and update their PIS.

2 Using Linked Open Data

User studies recently performed indicated new requirements, such as adding new services, changing their visualization, composing different services. In [4], prototypes developed to satisfy such requirements are presented. Moreover, a need for mashup data quality emerged. From the end users' point of view, data quality also refers to the amount of information provided by the retrieved services and its appropriateness to their needs [5]. Actually, some people using the platform prototype observed that few information were retrieved and with few details, compared to what they expected. This because end users are very diverse and have a great variety of needs. Often, not only a single service does not provide the required information, but even the mashup of more services is not capable to completely satisfy end users, who are increasingly more demanding.

I present here an approach that exploits linked data as a new source to be combined with services: it overcomes the above problem of data quality by adding semantic annotations to the services. Previous proposals tried to annotate services on the basis of manual or semi-automatic approaches [6]. The novelty of this approach is that it exploits automatic annotation of services, as described after the following scenario.

Scenario. Tony, the main persona of this scenario, is an organizer of entertainment events. He is looking for a musician for a local event he is organizing. He has added to his workspace *Grooveshark*, a service that, receiving in input the name of a musical artist, retrieves her albums and songs. Tony wants to know details like age, band, birthplace, etc. To find these details, Tony looks in the platform for some services to be combined with Grooveshark, but he is very disappointed because he does not find anything. How can linked data be useful to solve Tony's problem? The use of linked data enable Tony to query a knowledge base and have support in finding relevant data to be composed with other services.

The term linked data refers to best practices for publishing and linking structured data on the Web. Because they have been adopted by an increasing number of data providers, a global data space containing billions of assertions - the Web of Data – has been created. Linked data will be implemented in our platform by building a meta-layer over the registered services. In this layer, all relevant properties (input, output, etc.) of registered services are annotated by using classes' name of the ontologies adopted in the knowledge base of reference. This approach faces some critical issues: 1) identification of significant service properties to annotate; 2) how to label these properties, manually or automatically; 3) how to use these annotations to 'connect' services with linked data. For 1), being the considered mashups a composition of services oriented to information retrieval, the important properties are input query, output attributes and topic of service. For 2), a solution to automatically annotate services is the following. For each service offered by the platform, the platform administrator produces some significant query examples, which are used by an annotation engine to query these services and to collect many instances. Afterwards, for each service the automatic annotation phase starts: for each instance of each service attribute, the annotation engine queries a knowledge base, e.g. DBpedia, and obtains a set of classes related to the specific attribute instance. An algorithm establishes, for each attribute, the most important classes. The annotation engine annotates each attribute using the relevant classes. The same procedure is applied to the input attribute. This flow is summarized in this pseudo-code.

Annotation algorithm

```
     Input: Set T of Triples t=(s,A,Q), s is a service, A is a set of attributes a for s, and Q is a set of query on s
1:   for each t ∈ T do
2:      create I as empty set of instances results for queries
3:      for each q ∈ Q do
4:         query s by using q and collect instances results into I
5:      end for
6:      for each attribute a of s do
7:         create L as empty set of label for a
8:         for each i ∈ I do
9:            query DBpedia by using value of i respect to a and obtain a set C of classes
10:           put values of C into L
11:        end for
12:        annotate a by choosing most important class of L
13:     end for
14:  end for
```

Starting from the annotation of service attributes, an algorithm establishes the service topic. The proposed approach will be empowered by investigating recent algorithms and techniques proposed in ontology matching research area [7] and methods of semantic annotation of Web service based on DBpedia [8]. The result is a meta-level to each service that describes attributes in term of classes of an ontology. For example, in the case of our previous scenario, the artist_name attribute of Grooveshark is annotated with the Musical_artist class contained in DBpedia ontology. The most critical point is the third problem: how the system can exploit these annotations?

To cover the lack of information of previous scenario, the platform offers Tony the opportunity to use DBpedia as new data source. When Tony, starting from Grooveshark, opens DBpedia source in the mashup platform to add details, the system retrieves the classes used to annotate the artist_name attribute (Musical_artist in the previous example) and accesses to the DBpedia ontology to retrieve all possible links of the Musical_artist class with other classes or properties like age, instrument, style, etc. Tony can choose some of these links so that he can view in his future searches new artist's details contained in DBpedia.

This approach allows end users to deal linked data as new rich knowledge base connected with each service through an automatically built meta-layer transparent to the end user. However, this solution presents some open questions: how many classes have to be used to annotate services? How the system should choose the correct abstraction level of the class respect to the ontology? If the links of thighs to show to the users are too many, how the system should visualize all these information?

Acknowledgments. This work is partially supported by Italian Ministry of University and Research, grant VINCENTE, and by Ministry of Economic Development (MISE), grant LOGIN.

References

1. Casati, F.: How End-User Development Will Save Composition Technologies from Their Continuing Failures. In: Costabile, M.F., Dittrich, Y., Fischer, G., Piccinno, A. (eds.) IS-EUD 2011. LNCS, vol. 6654, pp. 4–6. Springer, Heidelberg (2011)
2. Namoun, A., Nestler, T., De Angeli, A.: Conceptual and Usability Issues in the Composable Web of Software Services. In: Daniel, F., Facca, F.M. (eds.) ICWE 2010. LNCS, vol. 6385, pp. 396–407. Springer, Heidelberg (2010)
3. Ardito, C., Costabile, M.F., Desolda, G., Lanzilotti, R., Matera, M., Piccinno, A., Picozzi, M.: User-driven visual composition of service-based interactive spaces. Journal of Visual Languages & Computing (in print)
4. Ardito, C., Costabile, M.F., Desolda, G., Lanzillotti, R., Matera, M., Picozzi, M.: Visual Composition of Data Sources by End-Users. In: Proc. of AVI 2014 (in print, 2014)
5. Picozzi, M., Rodolfi, M., Cappiello, C., Matera, M.: Quality-Based Recommendations for Mashup Composition. In: Daniel, F., Facca, F.M. (eds.) ICWE 2010. LNCS, vol. 6385, pp. 360–371. Springer, Heidelberg (2010)
6. Zeshan, F.: Semantic Web Service Composition Approaches: Overview and Limitations. Int. Journ. of New Computer Architectures and their Applications 3(1) (2011)
7. Shvaiko, P., Euzenat, J.: Ontology Matching: State of the Art and Future Challenges. IEEE Transactions on Knowledge and Data Engineering 25(1), 158–176 (2013)
8. Zhen, Z., Shizhan, C., Zhiyong, F.: Semantic Annotation for Web Services Based on DBpedia. In: IEEE Int. Symp. on Service Oriented System Eng., pp. 280–285 (2013)

Social Search

Marc Najork

Microsoft Research
1065 La Avenida, Mountain View, CA 94043, USA

Abstract. “Social Search” refers to two aspects of the integration of web search with social networks: how queries to a search engine may surface (socially) relevant content from social networks, and how signals from social networks may influence the (personalized) ranking of search results. The first part of the talk surveys the integration of Bing with Facebook, Twitter, Quora, Foursquare, LinkedIn, Klout, and other social platforms. The second part focuses on two technical details of this integration: a measure for quantifying the “affinity” between two users of a social network and an efficient algorithm for computing that measure, and a method for efficiently surfacing pages “liked” by your friends from a document-sharded index. The final part discusses limitations of social search, such as skewed demographics and weak homophily.

Keywords: Web search, social networks, social search.

1 Introduction

In this talk, I discuss the integration of web search with social networks. “Social Search” refers to two distinct aspects of this integration: how a user’s queries to a search engine may surface content from social networks (possibly authored by the searcher’s connections on that network), and how signals from social networks (say, the fact that a friend “liked” a web page) may influence the personalized ranking of algorithmic search results.

The talk is divided into three parts. In the first part, I survey the integration of Microsoft’s Bing Search engine with various social platforms, including Facebook, Twitter, Quora, Foursquare, LinkedIn, Klout, and others. Bing surfaces relevant content from multiple social networks, whether it is public or authored by the searcher’s connections on each network. It also promotes algorithmic search results that were endorsed by the user’s friends, or that are trending social media platforms.

Bing pays particular attention to “people search”, queries meant to retrieve relevant information about a person. Celebrity search is a well-studied problem, and retrieval precision is high, but this is less true for non-celebrity people search, due to the ambiguity of common names. Bing uses separation in social networks to surface individuals in the searcher’s extended social circle. Furthermore, it allows registered users to claim content related to them, thereby making it possible to cluster people search results by individual. Finally, it shows summaries of LinkedIn profiles directly on the results page. For celebrity searches, it will similarly show their presence in social media prominently on the results page together with related postings.

The second portion of the talk focuses on two technical details of Social Search. First, I describe an "affinity" measure [3] for quantifying how robustly connected two nodes in a graph (or two users of a social networks) are, or more precisely, what fraction of the graph's edges can be deleted before the nodes become disconnected. The affinity measure can be efficiently estimated by a randomized, sketch-based algorithm. The off-line phase of that algorithm computes a fixed-size sketch for each node of the graph, capturing a representative of its connected component at various levels of edge deletion. The online phase consists of retrieving the sketches of two nodes and performing a pointwise comparison on them to compute the affinity. The space complexity of the algorithm is $O(n)$, the time complexity of the off-line phase is $O(\alpha(n))$ (the complexity of union-find with path compression), while the time complexity of the online phase is $O(1)$.

Second, I discuss an approach for efficiently retrieving web pages "liked" by a user's friends. While seemingly trivial, it is challenging to integrate this functionality into a document-sharded distributed search index [2]. In such a setting, queries are distributed from a front-end to many index servers (each holding a part of the index), and results are sent back. Because of network constraints, both query and result transfers should be small; in particular, it is neither feasible to send the full set of the searcher's friends down the distribution tree, nor to send the full set of results up the aggregation tree. Moreover, social graphs can be very large and change continuously, making it impractical to maintain a copy of the graph on each index server.

The final part of the talk confesses to some of the limitations of Social Search; namely, that many social networks have skewed demographics [4] (in terms of gender, race, age, education and income), making it dangerous to generalize trends in networks to the overall population; and that while a user's actions on social networks *is* predictive of their proclivities [1], it is not clear that these preferences transfer to their "virtual" friends. Finally, it is challenging to "separate the wheat from the chafe" – to identify salient posts in a sea of the mundane.

References

1. Kosinski, M., Stillwell, D., Graepel, T.: Private traits and attributes are predictable from digital records of human behavior. Proceedings of the National Academy of Sciences, 5802–5805 (2013)
2. Najork, M.A., Panigrahy, R., Shenoy, R.K.: Considering document endorsements when processing queries. US Patent App. 13/218,450 (filed 2011)
3. Panigrahy, R., Najork, M., Xie, Y.: How user behavior is related to social affinity. In: 5th ACM International Conference on Web Search and Data Mining, pp. 713–722. ACM, New York (2012)
4. Rainie, L., Brenner, J., Purcell, K.: Photos and videos as social currency online. Pew Research Center (2012)

Wisdom of Crowds or Wisdom of a Few?

Ricardo Baeza-Yates

Yahoo Labs Barcelona
Barcelona, Spain

Abstract. In this keynote we focus on the concept of wisdom of crowds in the context of the Web, particularly through social media and web search usage. As expected from Zipf's principle of least effort, the wisdom is heterogeneous and biased to active people, which represent at the end the wisdom of a few. We also explore the impact on the wisdom of crowds of dimensions such as bias, privacy, scalability, and spam. We also cover related concepts such as the long tail of the special interests of people, or the digital desert, web content that nobody sees.

Summary

The Web continues to grow and evolve very fast, changing our daily lives. This activity represents the collaborative work of the millions of institutions and people that contribute content to the Web as well as more than two billion people that use it. In this ocean of hyperlinked data there is explicit and implicit information and knowledge. But how is the Web? What are the activities of people? What is the impact of these activities? Web data mining is the main approach to answer these questions. Web data comes in three main flavors: content (text, images, etc.), structure (hyperlinks) and usage (navigation, queries, etc.), implying different techniques such as text, graph or log mining. Each case reflects the wisdom of some group of people that can be used to make the Web better.

The wisdom of crowds [9] at work in the Web is best seen in social media as well as in social networks. It is also implicit in the usage of search engines [1] and other popular web applications. The wisdom behind web users is shaped by different complex factors such as the heterogeneity of user activity [10] and hence a heavy long tail [6]; different types of bias [3] that create problems such as the bubble effect [7]; privacy breaches coming from data [5]; too much data that endangers minorities [2]; or web spam in all possible ways [8].

The diversity of user activity implies that an elite of users represent most of the wisdom and that we should really talk about the wisdom of a few [4]. This diversity also generates a digital desert, web content that no one ever sees.

References

1. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval: The Concepts and Technology Behind Search, 2nd edn. Addison-Wesley (January 2011)
2. Baeza-Yates, R., Maarek, Y.: Usage Data in Web Search: Benefits and Limitations. In: Ailamaki, A., Bowers, S. (eds.) SSDBM 2012. LNCS, vol. 7338, pp. 495–506. Springer, Heidelberg (2012)

3. Baeza-Yates, R.: Big Data or Right Data? In: AMW 2013, Puebla, Mexico (May 2013)
4. Baeza-Yates, R., Saez-Trumper, D.: Wisdom of the Crowd or Wisdom of a Few? An Analysis of Users' Content Generation (submitted, 2014)
5. Barbaro, M., Zeller. Jr., T.: A face is exposed for AOL searcher no. 4417749. The New York Times (August 9, 2006)
6. Goel, S., Broder, A., Gabrilovich, E., Pang, B.: Anatomy of the long tail: ordinary people with extraordinary tastes. In: Proceedings of the Third ACM International Conference on Web Search and Data Mining, WSDM 2010, New York, NY, USA, pp. 201–210 (2010)
7. Pariser, E.: The Filter Bubble: What the Internet Is Hiding from You. Penguin Press (2011)
8. Spirin, N., Han, J.: Survey on web spam detection: principles and algorithms. ACM SIGKDD Explorations Newsletter Archive 13(2), 50–64 (2011)
9. Surowiecki, J.: The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations. Random House (2004)
10. Zipf, G.K.: Human behavior and the principle of least effort. Addison-Wesley Press (1949)

IFML: Building the Front-End of Web and Mobile Applications with OMG's Interaction Flow Modeling Language

Marco Brambilla

Politecnico di Milano. Dipartimento di Elettronica, Informazione e Bioingegneria
Piazza L. Da Vinci 32. I-20133 Milan, Italy
marco.brambilla@polimi.it

Abstract. Front-end design of Web applications is a complex and multidisciplinary task, where many perspectives intersect. A new standard modeling language called IFML (Interaction Flow Modeling Language) addresses this problem in a platform-independent way. IFML grants executability of models and binding to other aspects of system and enterprise design through integration with widespread software modeling languages such as UML, BPMN, SysML, SoaML and the whole MDA suite.

1 The UI Modeling Problem

In the last twenty years, capabilities such as form-based interaction, information browsing, link navigation, multimedia content fruition, and interface personalization have become mainstream and are implemented on top of a variety of technologies and platforms. However, no PIM-level design approach has emerged as a standard in the industry so far. Thus, front-end development continues to be a costly and inefficient process, where manual coding is predominant, reuse is low, and cross-platform portability is limited.

A possible solution to this problem is the Interaction Flow Modeling Language (IFML) [2], a visual notation for platform-independent design of software front end. IFML has been adopted as a standard by the Object Management Group (OMG) and features direct involvement of influential industrial players, seamless integration with widespread modeling languages such as UML, BPMN, SysML and the whole MDA suite, and availability of both open-source editors and industrial-strength implementations supporting end-to-end development.

2 The Interaction Flow Modeling Language (IFML)

The main contributions of IFML is the integration of best practices in the fields of model-driven development, software engineering, and Web engineering. IFML spawns mainly from the WebML [3] and the industrial experience obtained by the tool WebRatio [1]. IFML features user events and system events as first-class citizens, orthogonalization of business logic and interaction logic, and explicit and formal description of interoperability procedures and notations.

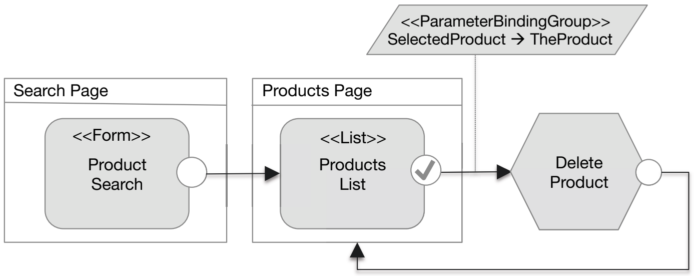


Fig. 1. IFML model example: search and listing of products, with deletion of an item

IFML supports the specification of: (1) *view structure*, i.e., visibility and reachability of view containers; (2) *view content*, , i.e., view components contained within view containers; (3) *user events and system events*; (4) *event transitions*, i.e., the effect of events on the user interface; (5) *parameter binding*, i.e., input-output dependencies between view elements; and (6) *binding to the business logic*, through references to application logic models and data models.

Figure 1 shows a simple example of IFML model where the user can search for products, gets the list of matching items and then can select one; the selection causes the selected product to be deleted and then leads back to the list.

Core Competencies. Besides learning the basics of the IFML syntax, the core competencies of IFML designers shall comprise: multi-perspective modeling, pattern-based design, UX-based design, and real-world industrial experience.

References

1. Acerbis, R., Bongio, A., Brambilla, M., Tisi, M., Ceri, S., Tosetti, E.: Developing ebusiness solutions with a model driven approach: The case of acer emea. In: Baresi, L., Fraternali, P., Houben, G.-J. (eds.) ICWE 2007. LNCS, vol. 4607, pp. 539–544. Springer, Heidelberg (2007)
2. Brambilla, M., Fraternali, P., et al.: The Interaction Flow Modeling Language (IFML), version 1.0 (2014), `http://www.ifml.org`
3. Ceri, S., Brambilla, M., Fraternali, P.: The History of WebML Lessons Learned from 10 Years of Model-Driven Development of Web Applications. In: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (eds.) Mylopoulos Festschrift. LNCS, vol. 5600, pp. 273–292. Springer, Heidelberg (2009)

Mashups: A Journey from Concepts and Models to the Quality of Applications

Cinzia Cappiello[1], Florian Daniel[2], and Maristella Matera[1]

[1] Politecnico di Milano
Via Ponzio 34/5, I-20133 Milano, Italy
{cinzia.cappiello,maristella.matera}@polimi.it
[2] University of Trento,
Via Sommarive 9, I-38123, Povo (TN), Italy
daniel@disi.unit.it

Abstract. This tutorial aims to provide insight into the constantly evolving mashup ecosystem. It presents core *definitions*, overviews a representative set of *mashup components* (the resources to be integrated) and *mashup models* (how resources are integrated), illustrates *composition paradigms* supported by state-of-the-art mashup tools, and discusses the *quality* of the resulting mashups from a user perspective. The goal of the tutorial is to introduce the topic and show its applicability, benefits and limitations.

1 Context and Motivation

The term "mashup" is widely used today. There are people developing "mobile mashups," others doing research on "Web mashups," and others again selling tools for "data mashups." Yet, when it comes to a concrete discussion of the topic, it is not uncommon to discover that the parties involved in the discussion actually have very different interpretations of what mashups are and what they are not. Typical discussion points are whether a mashup must have a user interface (UI) or not to be called a "mashup", whether it must be built by using Web-accessible resources only or not, whether it must be developed with client-side technologies (e.g., JavaScript) only, and the like. That is, even after several years that the term has been around and used, there is still no common agreement on its actual meaning and implications.

Interestingly, however, in the meantime mashing up data, functionalities and user interface widgets sourced from the Web has inexorably percolated into Web Engineering as a tacitly accepted development practice. Today, it is unimaginable to develop modern Web applications without some form of reuse and integration of value-adding, third-party content or services, a task that is greatly facilitated by technologies like Web services [3], the RESTful architectural style [2], Open Data, XML, JSON, W3C widgets, and many more.

But which are the conceptual underpinnings of this practice? What does it exactly mean to "mash up" resources that can be accessed via the Web? Which are the paradigms adopted for the composition of mashups? What kinds of tools exist that support this activity? And, eventually, what does it mean to develop "good" mashups? Working with students, discussing with colleagues, reading publications on the topic, we have seen that these questions are still open to many. We also identified a lack of suitable study material.

2 Learning Objectives

In light of these considerations, the learning objectives of this tutorial are:

- To obtain a basic understanding of the *core mashup aspects and concepts*, such as their contexts of use and target users, the most important definitions of mashups depending on the considered point of view (e.g., Web mashups, enterprise mashups, process mashups, telco mashups, mobile mashups, etc.), their intrinsic complexity and benefits.
- To get insight into the *most representative component technologies* used by mashups. Components are the basic elements of a mashup, and the comprehension of their characteristics and capability is fundamental for the understanding of what a mashup is and how it can be developed.
- To understand the *conceptual underpinning of mashups*, which developers must master and that can help them focus on the relevant issues when integrating components into mashups, as well as reference architectural patterns that can be instantiated. Both concepts and architectures can be analyzed independently of the particular technologies or sources used for an actual implementation.
- To gain insight into how mashup models and development practices can materialize into dedicated *mashup tools and composition paradigms* for assisted mashup development.
- To get insight into *quality models* for both mashup components and mashups, to understand how such models can guide the initial choice of components and the successive selection of composition patterns, and how they can also augment composition paradigms through the generation of quality-based recommendations.

The *target audience* are researchers, practitioners, advanced students who want to learn more about mashup development from a perspective that especially privileges abstractions and models, not only implementation aspects.

The tutorial is based on the authors' latest publication on mashups [1] and is complemented with an *online resource* providing additional material, slides and links for further study: `http://www.floriandaniel.it/mashupsbook`.

References

1. Daniel, F., Matera, M.: Mashups: Concepts, Models and Architectures. Springer (2014)
2. Fielding, R.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. Dissertation, University of California, Irvine (2007)
3. Papazoglou, M.P.: Web Services - Principles and Technology. Prentice Hall (2008)

Web of Things: Concepts, Technologies and Applications for Connecting Physical Objects to the Web

Iker Larizgoitia, Dominique Guinard, and Vlad Trifa

EVRYTHNG LIMITED, 4th Floor, 45 - 49 Leather Lane,
London EC1N 7TJ, United Kingdom
{iker,dom,vlad}@evrythng.com

Abstract. Inter-communicating devices and integrated Web-based services will open exciting opportunities. The idea of a world where everything is connected is becoming a reality, as hardware and software evolves to provide the necessary infrastructure to connect any physical device and objects to the Web. The goal of this tutorial is to present an overview on the advances in the Web of Things initiative. It will cover the key ideas behind the concept, how it works, infrastructure, as well as present the attendees with ideas and frameworks on how to build applications on top of it. The tutorial will have a practical approach and is open to both academia and industry attendees with interest in technologies and applications to interconnect physical objects to the Web. Knowledge of the architecture of the Web (e.g SOA, REST) and basic understanding of Web technologies (e.g. URIs, HTML, Javascript) are encouraged but not necessary to attend this tutorial.

1 Introduction

Inter-communicating devices and integrated Web-based services will open exciting opportunities. Smart appliances in smarter buildings will be linked in grids that can be more tightly monitored and regulated in real time. New communications protocols and software applications of fundamental importance are showing the way. Over the last several years, we have witnessed two major trends in the world of embedded devices:

- Hardware is becoming smaller, cheaper and more powerful, so that many devices will soon have communication and computation capabilities. Objects will be able to connect, interact and cooperate with other objects in their surrounding environment and with control centres—a vision generally dubbed the Internet of Things (IoT).
- The software industry is moving towards service-oriented integration technologies; especially in the business software domain, complex applications based on combined and collaborative services have been appearing. The Internet of Services (IoS) vision projects such integration on a large scale: services will reside in different layers of an enterprise; for example, in different operational units, IT networks or even run directly on devices and machines within a company.

As both of these trends are not domain-specific but common to multiple industries, we are facing a trend where the service-based information systems blur the border between the physical and virtual worlds, providing a fertile ground for a new breed of real-world aware applications. To facilitate these connections, research and industry have come up with a number of low-power network protocols such as Zigbee and Bluetooth, BLE and IPv6, in a version optimized for resource-constrained devices called 6lowpan.

Although they are increasingly part of our world, embedded devices still form multiple, small and incompatible islands at the application layer; developing applications to take advantage of them remains a very challenging task that requires expert knowledge of each platform. To ease the task, recent research initiatives have tried to provide uniform interfaces that create a loosely coupled ecosystem of services for smart things; these initiatives are often referred to as: "Web of Things".

2 Web of Things

The Web of Things is an evolution of the Internet of Things where the primary concern has been how to connect objects together at the network layer: similarly to the way the Internet addressed the lower-level connectivity of computers (layers 3-4 of the OSI model), the Internet of Things is primarily focusing on using various technologies such as RFID, Zigbee, Bluetooth or 6LoWPAN.

On the other hand, just like what the Web is to the Internet, the Web of Things regroups research and industrial initiatives looking into building an application layer for physical objects to foster their reusability and integration into innovative 3rd party applications. The envisioned approach is to reuse the already well-accepted and ubiquitous Web standards such as URI, HTTP, HTML5, REST, Web feeds, Javascript etc. Although these technologies were initially created for desktop computers, the fast increase of capabilities of embedded devices makes this possible already today.

3 Tutorial Objectives

In this tutorial we'll review a number of embedded devices and their integration to the Web, reviewing the Web of Things best practices on the way. We'll then dig into Web of Things platforms both on the commercial side, where we will present the EVRYTHNG platform and on the research side, where latest advances in object integration carried out in the COMPOSE platform will be presented. Based on these two platforms, use cases and applications will be showcased to demonstrate how to integrate objects, devices and applications following the Web of Things concepts.

Distributed User Interfaces and Multimodal Interaction

María D. Lozano[1], Jose A. Gallud[1], Víctor M.R. Penichet[1], Ricardo Tesoriero[1], Jean Vanderdonckt[2], Habib Fardoun[3], and Abdulfattah S. Mashat[3]

[1] Computing Systems Department, University of Castilla-La Mancha, Spain
{maria.lozano,victor.penichet,jose.gallud, ricardo.tesoriero}@uclm.es
[2] Université Catholique de Louvain, Belgium
jean.vanderdonckt@uclouvain.be
[3] King AbdulAziz University, Saudi Arabia
{hfardoun,asmashat}@kau.edu.sa

1 Objetives

This Workshop is the fourth in a series on Distributed User Interfaces. On this occasion, the workshop is focused on Distributed User Interfaces and Multimodal interaction. The main goal is to join together people working in extending the Web and other user interfaces to allow multiple modes of interaction such as GUI, TUI, Speech, Vision, Pen, Gestures, Haptic interfaces, etc. Multimodal interaction poses the challenge of transforming the way we interact with applications, creating new paradigms for developers and end-users. In a multi-device environment, we can find coupled displays, multi-touch devices, interactive table-tops, tablets, tangible user interfaces, eWatchs, etc., and this diversity of devices offers new possibilities and makes multimodal interaction even more challenging. Through active group discussion, participants will have the chance to share their knowledge and experience to advance in this field.

2 Theme and Topics

The main theme is to discuss and analyze how multimodal interaction can be applied in software applications based on Distributed User Interfaces (DUI). Moreover, interacting with the system through different means provides a richer user experience that it is worth exploring. Some of the topics to tackle are the following:

- Multimodal Interaction within Distributed User Interfaces
- Multimodal Web technologies
- Gesture-based interaction
- Multimodal Web Applications
- Multimodal mobile application
- Tangible user interfaces

3 Workshop Format

This one-day workshop will include a mix of paper presentations and breakout session activities according to the following scheme. With the goal of promoting debate among the workshop attendees, we will set the role of the “commentator” before the workshop, in such a way that every participant will be assigned the task of commenting the key points of other participant’s paper, promoting this way the discussion and participation of everybody.

During the workshop, we will divide it into two clearly differentiated parts. During the morning, participants will present their papers followed by short questions and discussion promoted by the “commentator”. Then, breakout sessions will be held during the afternoon. We will set working groups of 4-5 people to discuss about the topics of the workshop. This way we will promote active participation among all attendees to end up the workshop with useful conclusions and final remarks agreed by the attendees.

4 Expected Outcomes

We would like the workshop to be a forum to promote collaborations, generate ideas and synergies. We expect to find answers to the questions that would surely arise during the workshop and define the key guidelines that designers might consider when addressing the design of applications based on Distributed User Interfaces and the different ways of interacting within them. Besides, we also plan to organize a Special Issue in an International Journal with extended versions of a selection of papers, which will help to disseminate the workshop outcomes.

References

1. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target UI. Interacting with Computers 15(3)
2. Vandervelpen, C., Vanderhulst, G., Luyten, K., Coninx, K.: Light-weight Distributed Web Interfaces: Preparing the Web for Heterogeneous Environments. In: Lowe, D.G., Gaedke, M. (eds.) ICWE 2005. LNCS, vol. 3579, pp. 197–202. Springer, Heidelberg (2005)
3. Lozano, M.D., Gallud, J.A., Tesoriero, R., Penichet, V.M.R. (eds.): Distributed User Interfaces: Usability and Collaboration. Human–ComputerInteraction Series. Springer-Verlag London (2013) ISBN 978-1-4471-5498-3
4. Gallud, J.A., Tesoriero, R., Penichet, V.M.R. (eds.): Distributed User Interfaces. Designing Interfaces for the Distributed Ecosystem. Human-Computer Interaction Series. Springer-Verlag London (2011) ISBN: 978-1-4471-2270-8
5. Pedro, G., Villanueva, R., Tesoriero, J.A.: Gallud. Distributing web components in a display ecosystem using Proxywork. In: Proceedings of the 27th International BCS Human Computer Interaction Conference, BCS-HCI 2013, Article No. 28. British Computer Society, Swinton (2013)

Author Index