

**Tim A. Majchrzak
Paolo Traverso
Karl-Heinz Krempels
Valérie Monfort (Eds.)**

LNBIP 322

Web Information Systems and Technologies

**13th International Conference, WEBIST 2017
Porto, Portugal, April 25–27, 2017
Revised Selected Papers**

 **Springer**

Lecture Notes in Business Information Processing

322

Series Editors

Wil M. P. van der Aalst

RWTH Aachen University, Aachen, Germany

John Mylopoulos

University of Trento, Trento, Italy

Michael Rosemann

Queensland University of Technology, Brisbane, QLD, Australia

Michael J. Shaw

University of Illinois, Urbana-Champaign, IL, USA

Clemens Szyperski

Microsoft Research, Redmond, WA, USA

More information about this series at <http://www.springer.com/series/7911>

Tim A. Majchrzak · Paolo Traverso
Karl-Heinz Krempels · Valérie Monfort (Eds.)

Web Information Systems and Technologies

13th International Conference, WEBIST 2017
Porto, Portugal, April 25–27, 2017
Revised Selected Papers

Editors

Tim A. Majchrzak
University of Agder
Kristiansand, Vest-Agder Fylke
Norway

Paolo Traverso
Center for Information Technology
FBK-ICT irst
Trento
Italy

Karl-Heinz Krempels
RWTH Aachen University
Aachen
Germany

Valérie Monfort
University of Paris 1
Paris
France

ISSN 1865-1348 ISSN 1865-1356 (electronic)
Lecture Notes in Business Information Processing
ISBN 978-3-319-93526-3 ISBN 978-3-319-93527-0 (eBook)
<https://doi.org/10.1007/978-3-319-93527-0>

Library of Congress Control Number: 2018947335

© Springer International Publishing AG, part of Springer Nature 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by the registered company Springer International Publishing AG part of Springer Nature
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

The present book includes extended and revised versions of a set of selected papers from the 13th International Conference on Web Information Systems and Technologies (WEBIST 2017), held in Porto, Portugal, during April 25–27, 2017.

We received 77 paper submissions from 30 countries, of which 16% are included in this book. The papers were selected by the event chairs and their selection is based on a number of criteria that include the classifications and comments provided by the Program Committee members, the session chairs' assessment of the presentation and discussion quality, and also the program chairs' global view of all papers included in the technical program. The authors of selected papers were then invited to submit a revised and extended version of their papers having at least 30% novel material.

The purpose of the 13th International Conference on Web Information Systems and Technologies (WEBIST) was to bring together researchers, engineers, and practitioners interested in the technological advances and business applications of Web-based information systems. The conference had five main tracks, covering different aspects of Web information systems, namely: Internet Technology; Web Interfaces and Applications; Society, e-Communities; e-Business; Web Intelligence; and Mobile Information Systems.

The papers selected to be included in this book contribute to the understanding of relevant trends of current research on Web information systems and technologies, comprising:

- Unified interfaces
- Progressive Web apps (PWAs) as well as a mobile device taxonomy
- XML and open data processing
- The history of Web engineering
- Web development for end-users
- Access control
- Web platform assessment
- Rule engines
- Scientific blogging

We would like to thank all the authors for their contributions and also the reviewers who helped ensure the quality of this publication. Moreover, we wish to include a word of appreciation for the excellent organization provided by the conference secretariat run by INSTICC.

April 2017

Tim A. Majchrzak
Paolo Traverso
Karl-Heinz Krempels
Valérie Monfort

Organization

Conference Co-chairs

Karl-Heinz Krempels
Valérie Monfort

RWTH Aachen University, Germany
LAMIH Valenciennes UMR CNRS 8201, France

Program Co-chairs

Tim A. Majchrzak
Paolo Traverso

University of Agder, Norway
Center for Information Technology, IRST, Italy

Program Committee

Mohd Shahidan Abdullah
Jose Alfonso Aguilar
Jose Luis Herrero Agustin
Diana Andone
Marzal Miguel Ángel
Guglielmo De Angelis
Valeria De Antonellis
Giuliano Armano
Luca Berardinelli
Werner Beuschel
Adelaide Bianchini
Lidong Bing
Christoph Bussler
Elena Calude

Universiti Teknologi Malaysia, Malaysia
Universidad Autonoma de Sinaloa, Mexico
University of Extremadura, Spain
Politehnica University of Timisoara, Romania
Universidad Carlos Iii De Madrid, Spain
CNR, IASI, Italy
University of Brescia, Italy
University of Cagliari, Italy
TU Wien, Austria
Technische Hochschule Brandenburg, Germany
Universidad Simón Bolívar, Venezuela
Carnegie Mellon University, USA
Oracle Corporation, USA
Massey University, Institute of Natural
and Mathematical Sciences, New Zealand
University of Bari Aldo Moro, Italy
Politecnico di Milano, Italy
University of Turin, Italy
LUISS Guido Carli, Italy
Universitat Jaume I, Spain
Université Paris 8, France
The University of Hong Kong, SAR China
Ghent University, Belgium
Université Claude Bernard Lyon 1, France
University of South Wales, UK
University of Connecticut, USA
Università di Bologna, Italy
University of Malta, Malta

Pasquina Campanella
Cinzia Cappiello
Luigi Di Caro
Nunzio Casalino
Sven Casteleyn
Mario Cataldi
Dickson Chiu
Martine De Cock
Emmanuel Coquery
Daniel Cunliffe
Steven Demurjian
Enrico Denti
Alexiei Dingli

Martin Drlik	Constantine the Philosopher University in Nitra, Slovak Republic
Atilla Elci	Aksaray University, Turkey
Larbi Esmahi	Athabasca University, Canada
Jérôme Euzenat	Inria and University of Grenoble Alpes, France
Alexander Felfernig	Technische Universität Graz, Austria
Dieter A. Fensel	University of Innsbruck, Austria
Joao Carlos Amaro Ferreira	ISEL, Portugal
Josep-Lluís Ferrer-Gomila	Balearic Islands University, Spain
Karla Donato Fook	IFMA, Maranhão Federal Institute for Education, Science and Technology, Brazil
Geoffrey Charles Fox	Indiana University, USA
Pasi Fränti	University of Eastern Finland, Finland
Xiang Fu	Hofstra University, USA
Martin Gaedke	Chemnitz University of Technology, Germany
Ombretta Gaggi	Università di Padova, Italy
Yunjun Gao	Zhejiang University, China
John Garofalakis	University of Patras, Greece
Panagiotis Germanakos	University of Cyprus, Cyprus
Henrique Gil	Escola Superior de Educação do Instituto Politécnico de Castelo Branco, Portugal
Nuno Pina Gonçalves	Superior School of Technology, Polytechnic Institute of Setúbal, Portugal
Anna Goy	University of Turin, Italy
Carlos Granell	Universitat Jaume I, Spain
Ratvinder Grewal	Laurentian University, Canada
Daniela Grigori	Université Paris-Dauphine, France
Foteini Grivokostopoulou	University of Patras, Greece
Naijie Gu	University of Science and Technology of China, China
Karim El Guemhioui	Université du Québec en Outaouais, Canada
Angela Guercio	Kent State University, USA
Francesco Guerra	University of Modena and Reggio Emilia, Italy
Hakim Hacid	Zayed University, UAE
Fayçal Hamdi	Conservatoire National des Arts et Métiers, France
Shanmugasundaram Hariharan	Saveetha Engineering College, India
Ioannis Hatzilygeroudis	University of Patras, Greece
A. Henten	Aalborg University, Denmark
Hanno Hildmann	Universidad Carlos III de Madrid, Spain
Yuh-Jong Hu	National Chengchi University, Taiwan
Sergio Ilari	University of Zaragoza, Spain
Kai Jakobs	RWTH Aachen University, Germany
Anne James	Coventry University, UK
Monique Janneck	Lübeck University of Applied Sciences, Germany
Ivan Jelinek	Czech Technical University in Prague, Czech Republic
Zhuoren Jiang	Sun Yat-sen University, China

Ejub Kajan	State University of Novi Pazar, Serbia
Georgia Kapitsaki	University of Cyprus, Cyprus
George Karabatis	UMBC, USA
Ashraf Khalil	Abu Dhabi University, UAE
Matthias Klusch	German Research Center for Artificial Intelligence (DFKI) GmbH, Germany
In-Young Ko	Korea Advanced Institute of Science and Technology, South Korea
Waldemar W. Koczkodaj	Laurentian University, Canada
Hiroshi Koide	Kyushu University, Japan
Fotis Kokkoras	TEI of Thessaly, Greece
Efstratios Kontopoulos	Centre for Research and Technology Hellas, Greece
Tsvi Kuflik	The University of Haifa, Israel
Kin Fun Li	University of Victoria, Canada
Weigang Li	University of Brasilia, Brazil
Xian Li	LinkedIn Corp, USA
Dongxi Liu	CSIRO, Australia
Michael Mackay	Liverpool John Moores University, UK
Andrea Marrella	Università degli Studi di Roma La Sapienza, Italy
Kazutaka Maruyama	Meisei University, Japan
Wojciech Mazurczyk	Warsaw University of Technology, Poland
Luca Mazzola	Germany
Inmaculada Medina-Bulo	Universidad de Cádiz, Spain
Hakima Mellah	Research Center in Scientific and Technical Information, Algeria
Abdelkrim Meziane	CERIST Alger, Algeria
Alex Norta	Tallinn University of Technology, Estonia
Dusica Novakovic	London Metropolitan University, UK
Declan O'Sullivan	University of Dublin Trinity College, Ireland
Kalpdrum Passi	Laurentian University, Canada
David Paul	The University of New England, Australia
José António Sena Pereira	Instituto Politécnico de Setúbal, Escola Superior de Tecnologia de Setúbal, Portugal
Isidoros Perikos	University of Patras, Greece
Toon De Pessemier	Ghent University, iMinds, Belgium
Luis Ferreira Pires	University of Twente, The Netherlands
Simona Popa	Universidad Católica San Antonio de Murcia, Spain
Jim Prentzas	Democritus University of Thrace, Greece
Birgit Pröll	Johannes Kepler University Linz, Austria
Carne Quer	Universitat Politècnica de Catalunya, Spain
Thomas Risse	University Library Johann Christian Senckenberg, Germany
Andrzej Romanowski	Lodz University of Technology, Poland
Davide Rossi	University of Bologna, Italy
Gustavo Rossi	Lifia, Argentina
Yacine Sam	University of Tours, France

Comai Sara	Politecnico di Milano, Italy
Claudio Schifanella	Università degli Studi di Torino, Italy
Wieland Schwinger	Johannes Kepler University, Austria
Jochen Seitz	Technische Universität Ilmenau, Germany
Mohamed Sellami	RDI Group, LISITE Lab, ISEP Paris, France
Rami Sellami	CETIC, Belgium
Tacha Serif	Yeditepe University, Turkey
Xin Shuai	Thomson Reuters, USA
Marianna Sigala	University of South Australia Business School, Australia
Eliza Stefanova	Sofia University, Bulgaria
Dragan Stojanovic	University of Nis, Serbia
Dirk Thissen	RWTH Aachen University, Germany
Ismail Toroslu	Middle East Technical University, Turkey
Elena-Madalina Vatamanescu	National University of Political Studies and Public Administration, Romania
Jari Veijalainen	University of Jyväskylä, Finland
Maria Esther Vidal	Universidad Simon Bolivar, Venezuela
Petri Vuorimaa	Aalto University, Finland
Xinheng Wang	University of West London, UK
Tony Wasserman	Carnegie Mellon University, Silicon Valley, USA
Jason Whalley	Northumbria University, UK
Maarten Wijnants	Hasselt University, Belgium
Manuel Wimmer	Technische Universität Wien, Austria
Marco Winckler	University Nice Sophia Antipolis, France
William Van Woensel	Dalhousie University, Canada
Lina Zhou	University of Maryland, Baltimore County, USA

Additional Reviewers

Hassan Adelyar	Kabul University, Afghanistan
Alexandr Kormiltsyn	Tallinn University of Technology, Estonia
Karima Quayumi	Tallinn University, Estonia

Invited Speakers

Marco Gori	Università degli Studi di Siena, Italy
Christoph Rosenkranz	University of Cologne, Germany
Geert-Jan Houben	Delft University of Technology, The Netherlands
Roy Cecil	IBM Portugal, Portugal

Contents

TwigStackPrime: A Novel Twig Join Algorithm Based on Prime Numbers.	1
<i>Shtwai Alsubai and Siobhán North</i>	
Return of the Great Spaghetti Monster: Learnings from a Twelve-Year Adventure in Web Software Development	21
<i>Antero Taivalsaari and Tommi Mikkonen</i>	
Web Platform Assessment Tools: An Experimental Evaluation	45
<i>Solange Paz and Jorge Bernardino</i>	
Progressive Web Apps for the Unified Development of Mobile Applications	64
<i>Andreas Biørn-Hansen, Tim A. Majchrzak, and Tor-Morten Grønli</i>	
Web Performance Characteristics of HTTP/2 and Comparison to HTTP/1.1	87
<i>Robin Marx, Maarten Wijnants, Peter Quax, Axel Faes, and Wim Lamotte</i>	
CUBE System: A REST and RESTful Based Platform for Liquid Software Approaches.	115
<i>Clay Palmeira da Silva, Nizar Messai, Yacine Sam, and Thomas Devogele</i>	
Harnessing Community Knowledge in Heterogeneous Rule Engines	132
<i>Kennedy Kambona, Thierry Renaux, and Wolfgang De Meuter</i>	
Bringing Scientific Blogs to Digital Libraries: An Integration Process Workflow.	161
<i>Fidan Limani, Atif Latif, and Klaus Tochtermann</i>	
Enhanced Querying of Open Data Portals	179
<i>Mauro Pelucchi, Giuseppe Psaila, and Maurizio Toccu</i>	
A Taxonomy for App-Enabled Devices: Mastering the Mobile Device Jungle.	202
<i>Christoph Rieger and Tim A. Majchrzak</i>	
Attaining Role-Based, Mandatory, and Discretionary Access Control for Services by Intercepting API Calls in Mobile Systems	221
<i>Yaira K. Rivera Sánchez, Steven A. Demurjian, and Lukas Gnirke</i>	

Assisted End User Development for Non-programmers: Awareness,
Exploration and Explanation of Composite Web Application Functionality . . . 249
Carsten Radeck and Klaus Meißner

Author Index 277



TwigStackPrime: A Novel Twig Join Algorithm Based on Prime Numbers

Shtwai Alsubai¹(✉) and Siobhán North²

¹ College of Computer Engineering and Science,
Prince Sattam bin Abdulaziz University,
Al-Kharj, Kingdom of Saudi Arabia
sa.alsubai@psau.edu.sa

² Department of Computer Science,
The University of Sheffield, Sheffield, UK
s.north@sheffield.ac.uk

Abstract. The growing number of XML documents leads to the need for appropriate XML querying algorithms which are able to utilize the specific characteristics of XML documents. A labelling scheme is fundamental to processing XML queries efficiently. They are used to determine structural relationships between elements corresponding to query nodes in twig pattern queries (TPQs). This article presents a design and implementation of a new indexing technique which exploits the property of prime numbers to identify Parent-Child (P-C) relationships in TPQs during query evaluation. The Child Prime Label (CPL, for short) approach can be efficiently incorporated within the existing labelling schemes. Here, we propose a novel twig matching algorithm based on the well known TwigStack algorithm [3], which applies the CPL approach and focuses on reducing the overhead of storing useless elements and performing unnecessary join operations. Our performance evaluation demonstrates that the new algorithm significantly outperforms the previous approaches.

Keywords: XML databases · Holistic twig join algorithm
Node labelling · Twig pattern query

1 Introduction

As enterprises and businesses produce and exchange XML-formatted information more frequently, consequently, there is an growing requirement for effective handling of queries on data which conforms to an XML format [15–17]. Recently, several approaches have been proposed in the literature to process XML queries [3, 5, 7–12, 14, 17]. Due to the definition of relationships in XML as nested tags, data in XML documents are self-describing and flexibly organized [8, 16]. Therefore, the basic XML data model is a labelled and ordered tree.

In most XML query languages, such as XPath and XQuery, a twig (small tree) pattern can be represented as a node-labelled tree whose edges specify

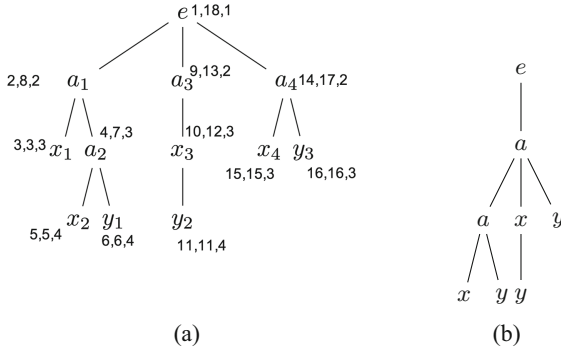


Fig. 1. (a) Range-based labelling scheme and (b) its *DataGuide*.

the relationship constraints among its nodes and they are either Parent-Child or Ancestor-Descendant. As a result, an XML query is defined as a complex selection on elements of an XML document specified by structural information of the selected elements. Improving the efficiency of tree patterns matching is a core operation in processing of an XML query [2, 4, 9, 10, 17] since tree patterns are the basis for querying structured tree-based data model such as XML.

Generally, the purpose of XML indexing is to improve the efficiency and scalability of query processing by reducing the search space. Without an index, XML retrieval algorithms have to scan all the data. Most existing XML query processing algorithms [5, 8, 9, 12, 20, 21] rely on XML indexing techniques to scan only the XML data relevant to XML queries, therefore, XML query performance is improved.

In XML, there are two basic type of indices. The first one is to index each node in an XML document by recording its positional information [13, 15]. This group is well-known as node label or labelling schemes. In this group of indices, every node in an XML document is assigned an unique label to record its position within the original XML document. The labelling scheme should enable determination of the structural information, i.e., Parent-Child (P-C) and Ancestor-Descendant (A-D) relationships. As a result, for any given two elements in an XML document, the relationship between them (if it exists) can be computed in constant time. A well-known example of node labelling is the containment labelling scheme proposed in [13]. In this approach, each node is assigned with a tuple of three values as $\langle start, end, level \rangle$. *Start* and *end* contain values of positions corresponding to the opening tag $\langle tag - name \rangle$ and the closing tag $\langle /tag - name \rangle$. *Level* represents the depth of the node within its XML tree. The two basic relationships Ancestor-Descendant and Parent-Child can be determined efficiently. Given two nodes u and v , u is an ancestor of v if and only if $u.start < v.start < v.end < u.end$. A Parent-Child relationship is defined as node u is the parent of node v if and only if $u.start < v.start < v.end < u.end$, $v.level = u.level + 1$. by way of explanation, the u node is in the range of node v .

The alternative to node labelling uses root-to-node paths in the XML document and is well-known as graph indexing (also referred to as structural summary or path indexing). Because an XML document can be modelled as rooted, ordered, labelled tree, a labelled path is defined as a sequence of tag names in the form of $tag_1/tag_2/\dots/tag_n$ from the root represented by tag_1 to node n tagged by tag_n . For illustration, consider the XML tree in Fig. 1, elements tagged by a can be stored in different storage structures according to their unique labelled paths. Consequently, elements corresponding to the path e/a are $\{a_1, a_3, a_4\}$, while a_2 is stored alone in its distinct labelled path $e/a/a$. A classic example of a path index is *DataGuide* [18]. The main drawback of this approach is that it only supports a simple path queries. There exist some graph indices that cover twig path queries as [19] but one of the limitations with these indices is that they are very large [5].

Both node and graph indexing are essential to XML query processing algorithms, they play important role in providing efficient evaluation of queries with respect to CPU complexity and memory consumption overhead [2, 21]. According to [22], a labelling scheme has to guarantee uniqueness and order preservation of node labels, thus the hierarchical relationships between a pair of nodes can be determined efficiently. The labelling scheme should enable checking all XPath relationships by computations only. To better understand the mechanisms of node indexing methods and their properties, [15] classified node indexing into four distinct types; range-based, prefix-based, multiplicative and hybrid labelling. A range-based labelling scheme will be adopted in this article to explore the effect of the new indexing mechanism. For sake of simplicity the following Example 1 aims to explain the use of labels in the determination of hierarchical relationships in XML trees.

Example 1. Consider Fig. 1, the structural relationships between the elements can be determined according to the properties for ancestor-descendant and parent-child relationships, respectively. Consider the relationship between node a_1 and y_1 , as the elements are labelled based on containment labelling scheme proposed in [13]. a_1 is an ancestor of y_1 because $2 < 6 < 8$. Also, a_1 is a parent node of a_2 because the parent-child conditions are satisfied as $2 < 4 < 8$ and $2 + 1 = 3$.

Organization. The rest of this article is organised as follows. Section 2 shows the related work. The new indexing technique will be introduced in Sect. 3. In Sect. 4, we present a holistic twig join matching algorithm *TwigStackPrime*. Section 5 presents thorough experimental studies about the performance between the new algorithm and the previous approaches. We conclude the paper in Sect. 6

2 Related Work

Every XML query processing algorithm which performs structural join operations to match a given query against an XML document relies on either range-based labelling schemes or prefix-based labelling schemes [3, 10, 14, 17]. This is

due to the fact that labelling schemes where nodes are considered as the basic unit of a query provides great flexibility in performing any structural query matching efficiently. The information gained from labels varies according to the chosen labelling scheme. To determine the effects of the range-based labelling scheme, [13] proposed multi-predict merge-join algorithm based on the positional information of the XML tree. An alternative representation, a prefix scheme, of labels of an XML tree can be seen in [10]. In this sort of labelling scheme, each node is associated with a sequence of integers that represents the node-ID path from the root to the node. This approach can be exemplified by the Dewey system used by librarians, the sequence of components in a Dewey label is separated by “.” where the last component is called the self label (i.e., the local order of the node) and the rest of the components are called the parent label. For instance, {1.2.3} is the parent of {1.2.3.1}. Another approach, [1] addressed the limitations of information encoded within labels produced by existing labelling schemes. It focus on performing join operations earlier, at leaf levels, where the selectivity of query nodes is at its peak for data-centric XML documents. The significance of the proposed approach stems from a comprehensive labelling scheme that encodes additional structural information, called *Nearest Common Ancestor*, *NCA for short* rather than the basic relationships among elements of XML documents. None of the previous approaches have taken the breadth of every node into account. In this paper, we propose a novel approach to overcome the previous limitations.

One of the most important problems in XML query processing is tree pattern matching. Generally, tree pattern matching is defined as mapping function M between a given tree pattern query Q and an XML document D , $M : Q \rightarrow D$ that maps nodes of Q into nodes of D where structural relationships are preserved and the predicates of Q are satisfied. Formally, tree pattern matching must find all matches of a given tree pattern query Q on an XML document D .

Early work on processing twig pattern matching decomposed twigs into a set of binary structures, then performed structural joins to obtain individual binary matchings. The final solution of the twig query is computed by stitching together the binary matches. In [3], the authors introduced the first holistic twig join algorithm for matching an XML twig pattern, called *TwigStack*. It works in two phases. Firstly, twig patterns are decomposed into a set of root-to-leaf paths queries and the solutions to these individual paths are computed from the data tree. Then, the intermediate paths are merge joined to form the final result. The authors of [3] proposed a novel prefix filtering technique to reduce the number of irrelevant elements in the intermediate paths.

The classical holistic twig join algorithm *TwigStack* only considers the ancestor-descendant relationship between query nodes to process a twig query efficiently without storing irrelevant paths in intermediate storage. It has been reported [3] that it has the worst-case I/O and CPU complexity when all edges in twigs are “//” (AD relationship) linear in the sum of the size of the input and output lists. However, *TwigStack*’s performance suffers from generating useless intermediate results when twig queries encounter Parent-Child relationships.

The authors of [9] proposed the first refined version of *TwigStack*. They introduced a new buffering technique to process twig queries with *P-C* relationships more efficiently by looking ahead some elements with *P-C* relationships in lists to eliminate redundant path solutions. *TwigStackList* guarantees every single path generated is a part of the final result if twig queries do not have *P-C* under branching query nodes. Subsequently, *TwigStackList* ensures optimal CPU and I/O cost when twig queries contain only Ancestor-Descendant edges below branching nodes and allows the occurrence of Parent-Child elsewhere [9]. The authors of [6] have proven that the *TwigStack* algorithm and its variants which depend on a single sequential scan of the input lists can not be optimal for evaluation of tree pattern query with any arbitrary combination of ancestor-descendant and parent-child relationships.

The approach to examine XML queries against document elements in post-order was first introduced by [4], *Twig²Stack*. The decomposition of twigs into a set of single paths and enumeration of these paths is not necessary to process twig pattern queries. The key idea of their approach is based on the proposition that when visiting document elements in post-order, it can then be determined whether or not they contribute to the final result before storing them in intermediate storage which is trees of stacks to ensure linear processing. *TwigList* [11] replaced the complex intermediate storage proposed in *Twig²Stack* with lists (one for every query node) and pointers with simple intervals to capture structural relationships. The authors in [7] proposed a new storage scheme, level vector split which splits the list connected to its parent list with *P-C* edge to a number of levels bounded by the maximum depth of the XML tree. A combination of pre-order and post-order filtering methods is adopted to develop two algorithms, namely: *TJStrictPre* and *TJStrictPost*. Although, they can prune irrelevant elements when P-C edges exist, they still perform unnecessary computations and store useless elements corresponding to leaf query nodes.

3 Child Prime Label

We present a new indexing technique which can be applied to the existing labelling schemes to skip scanning useless elements in the streams during the processing of twig pattern queries with Parent-Child edges. The key idea of our work is to find an appropriate, refined labelling scheme such that, for any given query node in the TPQ, the set of its child query nodes in the XML document, this forms the major bottleneck in determining structural relationship because Parent-Child can be resolved efficiently. This novel approach results in considerably fewer single paths stored than existing algorithm. It also increases the overall performance and reduces the memory overhead, and the result is shown clearly in our experiments.

The idea is to identify all the distinct tags in the XML tree and assign them with unique prime numbers. Then, the intuition of the CPL is to use the modulo function to create a mapping from an integer to a set of element names. The leaf elements will not be annotated with CPLs, whereas the inner elements (i.e.,

parent elements) are assigned CPLs. During depth-first scanning, an element is assigned the next available prime number if its tag has not been examined. After that, we check the CPL parameter of its parent element to see whether it is divisible by the assigned prime number or not. If it is, we process the next element, otherwise the product of parent element’s CPL is multiplied by the new prime number. We index tags for each XML tree in *tag indexing* to create a mapping from an element tag to a prime number as in Eq. 1. The tag indexing is implemented by a mean of *hash* table. For illustration, consider an element e , with all distinct names of children, $C = \{c_1, c_2, \dots, c_m\}$ and a list of prime numbers $P = \{p_1, p_2, \dots, p_n\}$. The bijective mapping function $f : C \rightarrow P$ for all element $p \in P$, there is a unique element $c \in C$ such that $f(c) = p$. Then, the CPL for element e can be computed as follows:

$$CPL(e) = \begin{cases} \prod_{i=1}^m f(c_i), & \text{if } m \geq 1 \\ \emptyset, & \text{otherwise} \end{cases} \quad (1)$$

Proposition 1 (Uniqueness). *There is only one unique set of prime factors for any number.*

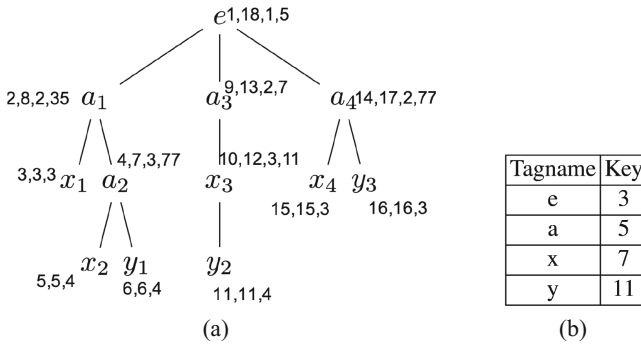


Fig. 2. (a) a sample of an XML tree labelled with the original range-based augmented with CPL parameters and (b) its corresponding tag indexing.

To explore the effect of CPL approach, we extend the original range-based labelling scheme to incorporate the CPL information. Each range-based label with CPL is presented as quadruple = $(start, end, level, CPL)$. The first three attributes remain the same as in the original labelling scheme see Sect. 1. According to Proposition 1, all distinct names of immediate child elements for a particular element in the XML tree can be obtained from having the corresponding prime numbers associated with tag names of its children.

Definition 1 (Child Prime Label). *A child prime label is assigned to each element in an XML document as an extra parameter into the range-based label.*

A child prime label indicates the multiplication of distinct prime numbers for every internal elements within the document. For example, node u is encoded quadruple = $(start_u, end_u, level_u, CPL_u)$.

Property 1. In any XML labelling scheme that is augmented with Child Prime Label, for any nodes x , y and z in an XML document, x has at least one or more child nodes of $tag(y)$ and $tag(z)$ if $CPL_x \bmod key_{tag}(y) \times key_{tag}(z) = 0$ where $key_{tag}(y)$ and $key_{tag}(z)$ are unique prime numbers.

To demonstrate the effect of child prime label, consider the XML tree in Fig. 2 and the tag indexing table on the top right, queries in XML are expressed as twigs since data is represented as tree. The answer to an XML query is all occurrences of it in an XML document under investigation. So, if we issue the simple twig query $Q = a[x]/y$, only two elements will be considered for further processing, namely a_2 and a_4 . This is because of $CPL_{a_2} \bmod key_{tag}(x) \times key_{tag}(y) = 77 \bmod 7 \times 11 \text{ equals } 0$.

4 Twig Join Algorithm

4.1 Notation

There is abstract data type called a stream, which is a set of elements with the same tag name, where the elements are sorted in ascending document order. Each query node q in a twig pattern is associated with an element stream, named T_q which has a cursor C_q which initially points to the first element in T_q at the beginning of a query processing. To ensure the linear processing in the filtering phase of holistic algorithms, only the first element is accessible and the rest of the elements are unseen by the algorithms. There are also some auxiliary operations on streams and TPQ and its nodes to facilitate the twig matching process. Supported operations are as follows: $getStart(C_q)$ returns the start attribute of the first element of q . $getEnd(C_q)$ returns the end attribute of the first element of q . $getLevel(C_q)$ returns the level attribute of the first element of q . $getCPL(C_q)$ returns the CPL attribute of the head element corresponding to query node q . $tagPrime(q)$ returns the unique prime number associated with q from *tag indexing*. $advance(C_q)$ forward the cursor of q to the next element. $eof(T_q)$ to judge whether C_q points to the end of stream of T_q . $children(q)$ returns all child nodes of q . $subtree(q)$ returns all child nodes which are in the subtree rooted at q . $childrenAD(q)$ returns all child nodes which have A-D relationship with q . $childrenPC(q)$ returns all child nodes which have P-C relationship with q . $isRoot(q)$ returns boolean values to see whether q is the root or not. $getRoot(TPQ)$ returns the query root of the input TPQ. $parent(q)$ returns the parent query node of q . $isLeaf(q)$ returns boolean values to see whether q is a leaf node or not.

4.2 TwigStackPrime

In this section, we present a new holistic twig join algorithm, called TwigStackPrime. It can be seen as an alternative to TwigStack algorithm. The structure

of the main algorithm, *TwigStackPrime* presented in Algorithm 2 is not much different from the original holistic twig join algorithm *TwigStack* [3] which uses two phases to compute answers to a TPQ. In the first phase, solutions to root-to-leaf paths in a TPQ are found and stored in output arrays (Lines 1–11). It repeatedly calls the *getNext* algorithm (see Algorithm 1) with the query root as the parameter to return the next query node for processing. In the second phase (Line 12), solutions in the output arrays are merge-joined based on their common branching query nodes and query matches are returned as the query result. The number of output arrays is equal to the number of leaf query nodes (i.e., the number of individual root-to-leaf paths in a TPQ).

Algorithm 1. getNext(q) [2].

Input: q is a query node
Result: a query node in TPQ which may or may not be q

```

1 if isLeaf(q) then
  | return: q
2 foreach node  $n_i$  in children(q) do
3   |  $g_i = \text{getNext}(n_i)$  if  $g_i \neq n_i$  then
  |   | return:  $g_i$ 
4  $n_{max} =$  a query node with the maximum start value  $\in$  children(q)
5  $n_{min} =$  a query node with the minimum start value  $\in$  children(q)
6 while  $\text{getEnd}(\text{getElement}(q)) < \text{getStart}(\text{getElement}(n_{max}))$  do
7   | advance(q)
8 if  $\text{getStart}(\text{getElement}(q)) < \text{getStart}(\text{getElement}(n_{min}))$  then
  | return: q
9 else
  | return:  $n_{min}$ 
10 Function getQCPL(Query node q):
11   | // the prime number assigned to the query node which is the product of its
  |   | child query node prime numbers
12   | qCPL = 1
13   | foreach node  $n_i$  in childrenPC(q) do
14   |   | qCPL = qCPL  $\times$  tagPrime( $n_i$ )
  |   | return: qCPL
15 Function getElement(Query node q):
16   | if childrenPC(q) > 0 then
17   |   | while  $\neg \text{eof}(C_q) \wedge \text{getCPL}(C_q) \% \text{getQCPL}(q) \neq 0$  do
18   |     | advance(q)
19   | if eof( $C_q$ ) then
  |     | return:  $\infty, \infty, \infty, 1$  // out of range label
20   |
21   | else
  |     | return:  $C_q$  // the current head element in the stream of q
22   |

```

getNext is a fundamental function which is called by the main algorithm to decide the next query node to be processed. It is used to guarantee that the

current element associated with the query node returned is part of the final output since all the basic structural relationships are thoroughly checked by *getNext* or its supporting subroutine *getElement*. *getNext(q)* returns an element e_q of a query node $q \in TPQ$ with three properties:

- i e_q has a descendant element e_{q_i} in each of the streams corresponding to its child elements where e_{q_i} is the first element of a query node $q_i = \text{children}(q)$ (this property is checked in Lines 9–11).
- ii each of its child elements satisfies recursively the first property (this property is checked in Lines 4–5).
- iii if q has Parent-Child edge(s) with its child query nodes, then e_q has a child e_{q_i} in T_{q_i} for each query node $q_i = \text{childrenPC}(q)$ (this property is checked in Lines 21–23 of *getElement* function).

In the function *getElement(q)*, if q does not have P-C edges, the first element of q is returned. Otherwise, Line 22 checks CPL relationship for all child query nodes with P-C relationships. If the first element does not satisfy the CPL relationship (the third property), the function skips all elements which do not satisfy the CPL relationship. Otherwise, the first element is found to satisfy the CPL relationship, then it is returned in Line 26 if the stream is unfinished. In case the stream reaches the end, Line 24 returns virtual end element labelled with infinity values as (∞, ∞, ∞) to complete the query processing.

Compared to the original *TwigStack* which does not apply CPL relationships, the effect of *TwigStackPrime* can be illustrated in the following example.

Example 2. Consider the XML tree of Fig. 3 and $Q_1 = a[//x]/y$. Assume the tree is labelled with range-based labelling and CPL approach as in Fig. 2. Initially, the cursors point at the first elements in streams. *getNext(a)* is called since a is the root query node. The first call of *getNext(a)* in *TwigStack* returns a_1 because it satisfies the descendant extension condition, but *TwigStackPrime* skips a_1 since it does not satisfy the CPL relationship that is CPL of a_1 is not divisible by the prime number associated with the tag name y . The algorithm has to skip n elements with tag x since they are useless to the first element a_2 . After this, *TwigStackPrime* can ensure that a_2 satisfies the three properties and thus is pushed into the stack for query node a . For instance, $\text{CPL}(a_1) \rightarrow 35$

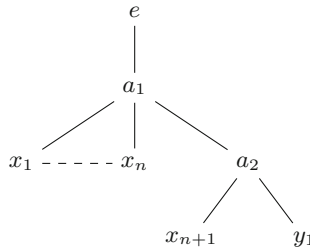


Fig. 3. Illustration to the suboptimal processing of *TwigStack*.

Algorithm 2. TwigStackPrime [2].

```

Input: TPQ Q
1 while  $\neg \text{end}(\text{getRoot}(Q))$  do
2    $q_{act} = \text{getNext}(\text{getRoot}(Q))$  // see Algorithm
3   if  $\neg \text{isRoot}(q)$  then
4      $\text{cleanStack}(\text{getElement}(q_{act}), \text{parent}(q_{act}))$ 
5   if  $\text{isRoot}(q) \vee \neg \text{empty}(S_{\text{parent}(q_{act})})$  then
6      $\text{cleanStack}(\text{getElement}(q_{act}), q_{act})$ 
7      $\text{moveToStack}(q_{act})$ 
8     if  $\text{isLeaf}(q_{act})$  then
9        $\text{outPathSolution}(q_{act})$  // Blocked solutions
10    else
11       $\text{advance}(q_{act})$ 
12  $\text{MergeAllPathSolutions}()$  // Phase 2
13 Function  $\text{cleanStack}(\text{Query node } q_{act}, \text{Query node } q)$ :
14   // pop any element in  $S_q$  which is not the ancestor of  $\text{getElement}(q_{act})$ 
15   while  $\neg \text{empty}(S_q) \wedge \text{getEnd}(\text{top}(S_q)) < \text{getStart}(\text{getElement}(q_{act}))$  do
16      $\text{pop}(S_q)$ 
17 Function  $\text{moveToStack}(\text{Query node } q)$ :
18   // p is a pointer to the top parent stack if q is the root p is null
19   //  $p = \text{top}(S_{\text{parent}(q)})$ 
20    $\text{push}(C_q, p)$  to  $S_q$ 
21 Function  $\text{end}(\text{Query node } q)$ :
   | return:  $\forall n_i \in \text{subtree}(q) : \text{isLeaf}(n_i) \wedge \text{eof}(C_{n_i})$ 

```

$\text{mod } \text{tagPrime}(y) \rightarrow 11$ is not equal to zero. The algorithm terminates after performing one recursive calls of $\text{getNext}(a)$. On the other hand, *TwigStack* has to iterate $n + 1$ times to answer match to Q_1 . *TwigStack* also generates n useless paths for Q_1 over the given XML tree.

4.3 Analysis of TwigStackPrime

In this section, we show the correctness of our algorithms. The correctness of *TwigStackPrime* algorithm can be shown analogously to *TwigStack* due to the fact that they both use the same stack mechanism. In other words, the correctness of Algorithm 2 follows from the correctness of *TwigStack* [3].

Definition 2 (Head Element). For each query node q in a TPQ Q , the element indicated by the cursor C_q is the head element of q .

Definition 3 (Child and Descendant Extension). A query node q has the child and descendant extension if the following properties hold:

- $\forall n_i \in \text{childrenAD}(q)$, there is an element e_i which is the head of T_{n_i} and a descendant of e_q which is the head of T_q .
- $\forall n_i \in \text{childrenPC}(q)$, there is an element e_q which is the head of T_q and its CPL parameter is divisible by $\text{tagPrime}(n_i)$.

– $\forall n_i \in \text{children}(q)$, n_i must have the child and descendant extension.

The above definition is a key for establishing the correctness of the following lemmas:

Lemma 1. *For any arbitrary query node q' which is returned by $\text{getNext}(q)$, the following properties hold:*

1. q' has the child and descendant extension.
2. Either $q == q'$ or q' violates the child and descendant extension of the head element e_q of its parent(q').

Proof. (Induction on the number of child and descendants of q). If q is a leaf query node, it is returned in Line 2 because it verifies all the properties 1 and 2a in Lemma 1. Otherwise, the algorithm recursively gets $g_i = \text{getNext}(n_i)$ for each child of q in Line 4. If for some i , there is $g_i \neq n_i$, and it is known by inductive hypothesis that g_i verifies the properties 1 and 2b with respect to q , so the algorithm returns g_i in Line 6. Otherwise, by inductive hypothesis that all q 's child nodes satisfy properties 1 and 2a with their corresponding sub-queries. At $\text{getElement}(q)$ (Lines 21–25), getNext advances from T_q all segments that do not satisfy the divisibility by the product of prime numbers in $\text{childrenPC}(q)$ returned from getQCPL . After that, the algorithm advances from T_q (Lines 9–10) all segments that are beyond the maximum start value of $n_i \in \text{children}(q)$. Then, if q satisfies properties 1 and 2a, it is returned at Line 12. Otherwise, Line 13 guarantees that $n_i \in \text{children}(q)$ with the smallest start value satisfies properties 1 and 2b with respect to start value of q 's head element e_q is returned.

Lemma 2. *Suppose $\text{getNext}(q)$ returns a query node q' and $q \neq q'$ at either Line 4 or 13 of getNext . Then there is no new solution involving top element of the parent stack of q' denoted as p which has end value less than the start value of the head element of q' or some elements which are in $\text{children}(p)$.*

Proof. Suppose that on contrary, there is a new solution using some elements of $p = \text{parent}(q')$ in S_p denoted as e_{S_p} for which $\text{getEnd}(e_{S_p}) < \text{getStart}(q')$. Using range-based property, it will be known that all elements from $\text{children}(p)$ in some solutions must have end values less than the end value of e_{S_p} , therefore less than the start value of the head element of q' . Since $\text{getNext}(q) = q'$ and from Line 3 of getNext for each child node n_i of p (including q'), it is $\text{getNext}(n_i) = n_i$ and $\text{getStart}(q') \leq \text{getStart}(n_i)$. Using Lemma 1, it will be known that each n_i has a child and descendant extension, and thus all elements of $\text{children}(n_i)$ have start values greater than $\text{getStart}(n_i)$, therefore greater than $\text{getStart}(q')$, which is a contradiction.

Theorem 1. *Given a twig pattern query Q and an XML document D , Algorithm *TwigStackPrime* correctly returns answer to Q on D .*

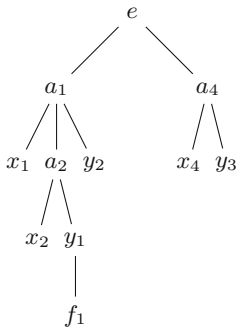
Proof. In Algorithm *TwigStackPrime*, $\text{getNext}(\text{root})$ is repeatedly invoked to determine the next query node to be processed. Using Lemma 1, it is known

that all elements returned by $q_{act} = getNext(root)$ have the child and descendant extension. If $q_{act} \neq root$, Line 4, the algorithm pops from $S_{parent(q_{act})}$ all elements that are not ancestors of the head element of q_{act} by Lemma 2. After that, it is already known q_{act} has a child and descendant extension so that Line 5 checks whether $S_{parent(q_{act})}$ is empty or not. If so, it indicates that it does not have the ancestor extension, and it can be discarded safely to continue with the next iteration. Otherwise, the current head element of q_{act} has both the ancestor and child and descendant extensions which guarantee its participation in at least one root-to-leaf path. Then, $S_{q_{act}}$ is cleaned by popping elements which do not contain the head of q_{act} . Then, the item in the stack is used to maintain pointers from itself to the query root. Finally, if q_{act} is a leaf node, we compute all possible combinations of single paths with respect to q_{act} , line 8–9.

The correctness holds for TPQs with both A-D and P-C relationships, it can be shown that *TwigStackPrime* algorithm is optimal when P-C axes exist only in the deepest level of a twig query. The intuition is simple since the CPL relationship can detect hidden immediate child elements only in two streams related by P-C relationships. Henceforth, we can conclude the following result.

Theorem 2. Consider a twig pattern query Q with n query nodes, and only Ancestor-Descendant edges or there are Parent-Child edges to connect leaf query nodes, and an XML document D . *TwigStackPrime* has worst-case I/O and CPU time complexities linear in the sum of the size of the n input lists and the output list.

Example 3. Consider the XML tree of Fig. 4 and $Q_2 = a[/x]/y/f$, the head elements in their streams are $a \rightarrow a_1$, $x \rightarrow x_1$, $y \rightarrow y_1$ and $f \rightarrow f_1$. The first call of $getNext(root)$ inside the main algorithm will return $a \rightarrow a_1$ because it has A-D relationship with all head elements and satisfies CPL with x and y , and its



(a) an XML tree.

Tagname	Key
e	3
a	5
x	7
y	11
f	13

(b) tag indexing.

Fig. 4. Sub-optimal evaluation of *TwigStackPrime* where redundant paths might be generated.

descendant $y \rightarrow y_1$ also satisfies the child and descendant extension with respect to f . However, *TwigStackPrime* produces the useless path (a_1, x_1) because y_2 does not have child of f -node [2].

5 Experimental Evaluation

In this section we present the performance comparison of twig join algorithms, namely: *TwigStackPrime* the new algorithm based on *Child Prime Label* approach, along with *TwigStack* [3]. The original twig join algorithm that was reported to have optimal worst-case processing with A-D relationship in all edges, and *TwigStackList* is the first refined version of *TwigStack* to process P-C efficiently [9]. *TwigStackList* was chosen in this experiment because it utilizes a simple buffering technique to prune irrelevant elements from streams. We evaluated the performance of these algorithms against both real-world and artificial datasets. The performance comparison of these algorithms was based on the following metrics:

1. Number of intermediate solutions: the individual root-to-leaf paths generated by each algorithm.
2. Processing time: the main-memory running time without counting I/O costs. All twig pattern queries were executed 103 times and the first three runs were excluded for cold cache issues. We did not count the I/O cost for tag indexing files for *TwigStackPrime* algorithm because it is negligible, and the cost to read the tag indexing is constant over a series of twig pattern queries.

5.1 Experimental Settings

All the algorithms were implemented in Java JDK 1.8. The experiments were performed on 2.9 GHz Intel Core i5 with 8 GB RAM running in Mac OS X El Capitan. The benchmarked data sets used in the experiments and their characteristics are shown in Table 1. The selected datasets and benchmark are significantly more frequent in the literature of XML query processing [3, 7–9, 11, 12].

Table 1. Characteristics of XML datasets used in the experiments.

	DBLP	TreeBank	XMark	Random
Range-based MB	65.3	43	35.3	69.4
CPL MB	70.3	47.9	40.1	74.1
Δ size MB	5	4.9	4.8	4.7
Tag indexing size KB	0.48	3	1	0.049
Nodes (Millions)	3.73	2.43	2.04	3.94
Max/Avg depth	6/2.9	36/7.8	12/5.5	13/7
Distinct tags	40	251	83	6
Largest prime numbers	151	1597	379	19

DBLP is a highly structured document and is very wide and shallow, while TreeBank is a deep-recursive dataset with a large number of distinct tags and has irregular structure. Both are real-world and obtained from the University of Washington XML repository [24]. The XMark dataset is well-known benchmarked XML dataset [23]. To ensure fair comparison, DBLP and XMark datasets were selected because they are both considered as data-oriented and have very strong structures. We also generated Random dataset similar to that in [9] but we have the two parameters: *depth* and *fan-out*. The depth of randomly generated tree has maximum value sets to 13 and *fan-out* has range from 0 to 6, respectively. This dataset was created to test the performance where the

Table 2. Experimental TPQs.

Code	XPath expression	Result
DQ_1	/dblp/inproceedings[//title//author	88
DQ_2	//www[editor]/url	21
DQ_3	//article[//sup//title//sub	278
DQ_4	//article[/sup//title/sub	0
XQ_1	/site/closed_auctions/closed_auction[annotation/description/text/keyword]/date	4042
XQ_2	/site/closed_auctions/closed_auction//keyword	12527
XQ_3	/site/closed_auctions/closed_auction[/keyword]/date	12527
XQ_4	/site/people/person[profile[gender][age]]/name	3243
XQ_5	//item[location][//mailbox//mail//emph]/description//keyword	16956
XQ_6	//people/person[/address/zipcode]/profile/education	3241
TQ_1	//S[//MD]//ADJ	19
TQ_2	//S/VP/PP[/NP/VBN]/IN	152
TQ_3	//VP[/DT]//PRP_DOLLAR_	3
TQ_4	//S[/JJ]/NP	5
TQ_5	//S[VP[DT]//NN]/NP	32
TQ_6	//S[/VP/IN]//NP	20311
TQ_7	//S/VP/PP[/NP/VBN]/IN	320
TQ_8	//EMPTY/S//NP[/SBAR/WHNP/PP//NN]/_COMMA_	17
TQ_9	//SINV//NP[/PP[/JJR][//S]/NN	4
RQ_1	//b//e//a[/f][d]	1331
RQ_2	//a//b[/e][c]	18033
RQ_3	//e//a[/b][c]	11216
RQ_4	//a[/b/d]//c	59568
RQ_5	//b[d/f]/c[e]/a	377
RQ_6	//c[/b][a]/f	47159
RQ_7	//a[c//e]/f[d]	1906
RQ_8	//d[a//e/f]/c[b]	204
RQ_9	//a[d][c][b][e]//f	3757

XML document combines features of DBLP and TreeBank, being structured and deeply-recursive at the same time.

The XML structured queries for evaluation over these dataset were chosen specifically because it is not common for queries, which contain both ‘//’ and ‘/’, to have a significant difference in performance for tightly-structured document such as DBLP and XMark. TreeBank twig queries were obtained from [7,9]. Twig patten queries over the random data set were also randomly generated. Table 2 shows the XPath expressions for the chosen twig patterns. The code indicates the data set and its twig query, for instance, TQ2 refers to the second query issued over TreeBank dataset.

5.2 Experimental Result

We compared *TwigStackPrime* algorithm with *TwigStack* and *TwigStackList* over the above mentioned twig pattern queries against the data sets selected. The Kruskal-Wallis test is a non-parametric statistical procedure was carried out on processing time, the *p-value* turns out to be nearly zero (p-value less than 2.2^{-16}), it strongly suggests that there is a difference in processing time between two algorithms at least as shown in Fig. 5.

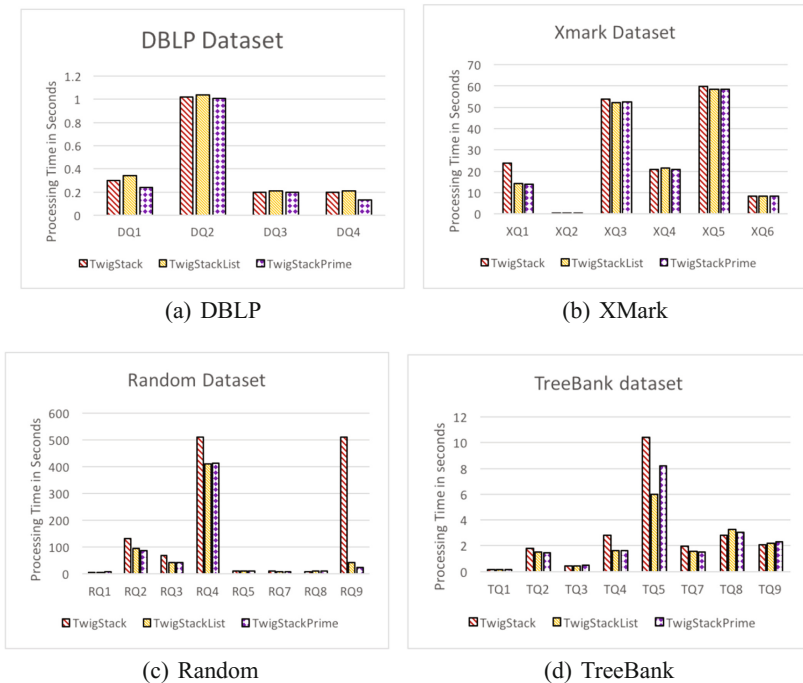


Fig. 5. Processing time for twig pattern queries against DBLP in (a) and XMark in (b). (c) and (d) shows processing time for TPQs on Random and TreeBank datasets, respectively [2].

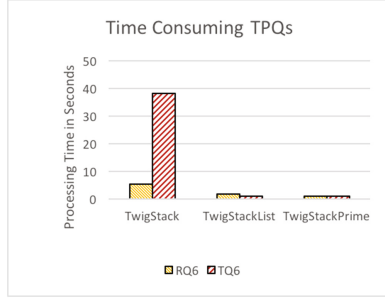


Fig. 6. The processing time taken by each algorithm to run the two most expensive queries in the experiments, normalizing query times to 1 for the fastest algorithm for each query [2].

TwigStackPrime vs. TwigStack. We compare the performance between *TwigStackPrime* and *TwigStack*. Table 3 shows that *TwigStackPrime* always generates fewer root-to-leaf paths than *TwigStack*. This is because *TwigStackPrime* uses CPL relationships to prune irrelevant elements. For instance, in TQ_6 , *TwigStackPrime* produced only 22,565 useful paths, whereas the number of intermediate paths in *TwigStack* was 702,391. Although DBLP and XMark have relatively regular structures, *TwigStack* still produced irrelevant paths. For this type of datasets, *TwigStackPrime* shows optimal performance by generating only paths contributing in the final results. Since there is a difference in performance suggested by the Kruskal-Wallis test, we ran pairwise comparison based on Mann-Whitney test which showed that in most test twig queries *TwigStackPrime* outperformed *TwigStack* as depicted in Fig. 5. In our experiments, we used TQ_6 and RQ_6 because they touch very large portions of their datasets and produce quite huge results. For TQ_6 and RQ_6 , *TwigStackPrime* were more than 40 and 5 time faster than *TwigStack*, respectively.

TwigStackPrime vs. TwigStackList. We now compare the performance between *TwigStackPrime* and *TwigStackList*. For highly structured datasets, both *TwigStackPrime* and *TwigStackList* are optimal as presented in Table 3. However, none of the algorithms are optimal in the other datasets because they have redundant paths and many tags are deeply recursive. In most queries, *TwigStackPrime* generated relatively fewer paths than *TwigStackList*. This is because *TwigStackPrime* uses CPL relationships to prune useless elements while *TwigStackList* utilises a simple buffering technique bounded by the number of elements in the longest path of the queried XML dataset. For example, RQ_9 where some of branching edges are P-C, *TwigStackPrime* can guarantee optimal evaluation because RQ_9 is its optimal class of query as mentioned in Theorem 2. *TwigStackPrime* produced 8,786 useful paths whereas *TwigStackList* generated 17,328 useless paths. Even though RQ_4 is optimal for *TwigStackList* because it

Table 3. Single paths produced by each algorithm [2].

Code	TwigStack	TwigStack list	TwigStackPrime
DQ_1	147	139	139
DQ_4	98	0	0
XQ_1	9414	6701	6701
TQ_2	2236	388	441
TQ_3	10663	11	5
TQ_4	70988	30	10
TQ_6	702391	22565	22565
TQ_8	58	27	26
TQ_9	29	17	8
RQ_1	2076	1843	1795
RQ_2	29914	24235	23057
RQ_3	20558	16102	15505
RQ_4	67005	57753	57753
RQ_5	3765	901	1093
RQ_6	201835	98600	72084
RQ_7	6880	2791	3219
RQ_8	746	322	406
RQ_9	179546	26114	8786

does not have P-C in branching axes, *TwigStackPrime* evaluated RQ_4 efficiently see Fig. 5 and Table 3.

Since there was a difference in performance, we ran pairwise comparison based on Manny-Whitney test which showed that in most twig queries tested *TwigStackPrime* outperformed *TwigStackList*, however, they showed same performance in XQ_2 , XQ_3 and XQ_6 see Fig. 5. For expensive queries, pairwise comparison based on Manny-Whitney test between *TwigStackPrime* and *TwigStackList* resulted in p -value < 0.001 which suggests a significant difference and *TwigStackPrime* has the best performance.

When evaluating RQ_6 , *TwigStackPrime* has the best performance, it is roughly twice as fast than *TwigStackList*.

Summary. It can be seen in Fig. 5 the only twig queries where *TwigStackPrime* has slower performance comparing to the others is TQ_3 and TQ_9 because they touch very little of the dataset. According to the experimental results, we can draw the following two conclusions:

1. The CPL approach is a new source of improvement for holistic twig matching algorithms since it can reduce the number of elements processed and the size of intermediate result when TPQs contain Parent-Child edges.

2. *TwigStackPrime* significantly outperformed *TwigStack* and *TwigStackList* for different types of XML documents in terms of their structures including shallow and deep datasets. *TwigStackPrime* showed a superior performance in avoiding the storage of unnecessary paths while processing time is improved.

6 Conclusion and Future Work

In this paper, we proposed the CPL approach to improve the pre-filtering strategy in twig join algorithms when *P-C* edges are involved in TPQs. The key to the *TwigStackPrime* is the use of the CPL approach as the labelling scheme and of the advanced preorder filtering function *getNext*, which both enable fast determination of P-C relationships between elements of XML documents while scanning them in preorder traversal. This property is exploited to reduce storage space by skipping irrelevant elements from the streams and to improve the overall performance.

Compared to the previous labelling schemes, the CPL approach can be used to derive a set of the tag names of child elements associated with their inner elements. P-C edges, hence, can be solved in very efficient way. *TwigStackPrime* algorithm shows the general framework we use for introducing the CPL approach into existing twig matching algorithms, extending algorithm like *TwigStack*.

Existing research revolves around improving the efficiency of twig matching algorithms and extending querying algorithms to make them more able to handle positional predicates and order axes in XPath expressions. A promising approach for speeding up the query processing would be to combine our approach with the previous orthogonal algorithms to propose a new one-phase twig matching algorithm that we hope will be superior to the previous approaches. The current preliminary idea is to examine processing ordered twig patterns and positional predicate in a way that would consume less time and memory than the existing approaches. We will consider one-phase and ordered twig matching algorithms as our future work.

References

1. Aghili, S.A., Li, H.-G., Agrawal, D., El Abbadi, A.: TWIX: twig structure and content matching of selective queries using. In: Proceedings of the 1st International Conference on InfoScale 2006, p. 42 (2006)
2. Alsubai, S., North, S.: A prime number approach to matching an XML twig pattern including parent-child edges. In: The 13th International Conference on Web Information Systems and Technologies, WEBIST 2017, pp. 204–211. SCITEPRESS Science and Technology Publications, Lda, Porto (2017)
3. Bruno, N., Koudas, N., Srivastava, D.: Holistic twig joins: optimal XML pattern matching. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, pp. 310–321. ACM, Madisonn (2002)
4. Chen, S., Li, H.-G., Tatemura, J., Hsiung, W.-P., Agrawal, D., Sel, K., Candan, K.S.: Twig2Stack: bottom-up processing of generalized-tree-pattern queries over XML documents (2006)

5. Chen, T., Lu, J., Ling, T.W.: On boosting holism in XML twig pattern matching using structural indexing techniques. In: Science, pp. 455–466 (2005)
6. Choi, B., Mahoui, M., Wood, D.: On the optimality of holistic algorithms for twig queries. In: Mařík, V., Retschitzegger, W., Štěpánková, O. (eds.) DEXA 2003. LNCS, vol. 2736, pp. 28–37. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45227-0_4
7. Grimsmo, N., Bjørklund, T.A., Hetland, M.L.: Fast optimal twig joins. VLDB **3**(1–2), 894–905 (2010)
8. Li, J., Wang, J.: Fast matching of twig patterns. In: Bhowmick, S.S., Küng, J., Wagner, R. (eds.) DEXA 2008. LNCS, vol. 5181, pp. 523–536. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85654-2_45
9. Lu, J., Chen, T., Ling, T.W.: Efficient processing of XML twig patterns with parent child edges: a look-ahead approach. In: Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management, no. i, pp. 533–542. ACM, Washington, D.C. (2004)
10. Jiaheng, L., Meng, X., Ling, T.W.: Indexing and querying XML using extended Dewey labeling scheme. Data Knowl. Eng. **70**(1), 35–59 (2011)
11. Qin, L., Yu, J.X., Ding, B.: *TwigList*: make twig pattern matching fast. In: Kotagiri, R., Krishna, P.R., Mohania, M., Nantajeewarawat, E. (eds.) DASFAA 2007. LNCS, vol. 4443, pp. 850–862. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71703-4_70
12. Wu, H., Lin, C., Ling, T.W., Lu, J.: Processing XML twig pattern query with wildcards. In: Liddle, S.W., Schewe, K.-D., Tjoa, A.M., Zhou, X. (eds.) DEXA 2012. LNCS, vol. 7446, pp. 326–341. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32600-4_24
13. Zhang, C., Naughton, J., DeWitt, D., Luo, Q., Lohman, G.: On supporting containment queries in relational database management systems. ACM SIGMOD Rec. **30**, 425–436 (2001)
14. Al-Khalifa, S., Jagadish, H.V., Koudas, N., Patel, J.M., Srivastava, D., Wu, Y.: Structural joins: a primitive for efficient XML query pattern matching. In: Proceedings of 18th International Conference on Data Engineering, pp. 141–152 (2002)
15. Haw, S.-C., Lee, C.-S.: Data storage practices and query processing in XML databases: a survey. Knowl. Based Syst. **24**(8), 1317–1340 (2011)
16. Gang, G., Chirkova, R.: Efficiently querying large XML data repositories: a survey. IEEE Trans. Knowl. Data Eng. **19**(10), 1381–1403 (2007)
17. Lu, J., Ling, T.W., Bao, Z., Wang, C.: Extended XML tree pattern matching: theories and algorithms. IEEE Trans. Knowl. Data Eng. **23**(3), 402–416 (2011a)
18. Goldman, R., Widom, J.: Dataguides: enabling query formulation and optimization in semistructured databases. In: Proceedings of International Conference on Very Large Data Bases, pp. 436–445 (1997)
19. Kaushik, R., Bohannon, P., Naughton, J.F., Korth, H.F.: Covering indexes for branching path queries. In: Proceedings of 2002 ACM SIGMOD International Conference on Management Data, SIGMOD 2002, p. 133 (2002)
20. Bača, R., Krátký, M., Ling, T.W., Lu, J.: Optimal and efficient generalized twig pattern processing: a combination of preorder and postorder filterings. VLDB J. **22**(3), 369–393 (2012)
21. Bača, R., Krátký, M.: XML query processing. In: Proceedings of 16th International Database Engineering Application Symposium, IDEAS 2012, p. 813 (2012)
22. Mathis, C., Härder, T., Schmidt, K., Bächle, S.: XML indexing and storage: fulfilling the wish list. Comput. Sci. Res. Dev. **30**, 118 (2012)

23. Schmidt, A., Waas, F., Kersten, M., Busse, R., Carey, M.J., Amsterdam, G.B.: XMark: a benchmark for XML data management. In: VLDB 2002 Proceedings of the 28th International Conference on Very Large Data Bases, pp. 974–985 (2002)
24. Miklau, G.: UW XMLData Repository. <http://www.cs.washington.edu/research/xmldatasets/>. Accessed 04 Feb 2016



Return of the Great Spaghetti Monster: Learnings from a Twelve-Year Adventure in Web Software Development

Antero Taivalsaari¹ and Tommi Mikkonen²(✉)

¹ Nokia Technologies, Tampere, Finland
`antero.taivalsaari@nokia.com`

² University of Helsinki, Helsinki, Finland
`tommi.mikkonen@helsinki.fi`

Abstract. The widespread adoption of the World Wide Web has fundamentally changed the landscape of software development. Only ten years ago, very few developers would write software for the Web, let alone consider using JavaScript or other web technologies for writing any serious software applications. In this paper, we reflect upon a twelve-year adventure in web development that began with the development of the *Lively Kernel* system at Sun Microsystems Labs in 2006. Back then, we also published some papers that identified important challenges in web-based software development based on established software engineering principles. We will revisit our earlier findings and compare the state of the art in web development today to our earlier learnings, followed by some reflections and suggestions for the road forward.

Keywords: Web programming · Web applications · Web engineering
Software engineering · HTML5 · JavaScript · The Internet of Things
IoT · Programmable World

1 Introduction

The widespread adoption of the World Wide Web has fundamentally changed the landscape of software development. In the past years, the Web has become the *de facto* deployment environment for new software systems and applications. Office productivity applications and corporate tools such as invoicing, purchasing and expense reporting systems have migrated to the Web. Banking, insurance and retail industries – to name a few – have been transformed profoundly by the emergence of web-based applications and internet services. Academic papers such as this one are now commonly written using collaborative, browser-based environments instead of traditional, installed office suites. Even software development is nowadays often performed using interactive, web-based tools.

Over ten years ago, we published a number of papers on the emergence of the Web as a software development platform and associated challenges [1, 2].

At that point, the world looked very different still. Back in 2006, very few developers would write software for the Web, let alone consider using JavaScript or other web technologies for writing any serious software applications [3]. Today, the Software as a Service (SaaS) model [4] is prevalent, and interactive, dynamic software development for the Web has become commonplace. In fact, traditional installed applications now maintain a stronghold only in the mobile realm, where the number of mobile apps (especially for iOS and Android devices) has exploded in recent years [5]. In contrast, the number of applications that people install on their personal computers has been in steady decline over the past years. The majority of activities on personal computers are now performed using a web browser, leveraging the Software as a Service model [6].

A key technical manifestation of the early years of our twelve-year adventure in web development was the *Lively Kernel* system (<http://lively-kernel.org/>), originally created at Sun Microsystems Labs in 2006–2008. The Lively Kernel was one of the first fully interactive, self-sustaining, web-based software development environment that was built on the assumption that the web browser would become a credible, full-fledged software platform [7]. While the Lively Kernel is not very widely known or used today, it did pave the way – for its part – for today’s Software as a Service based software development systems and truly interactive, live web programming. A recently published ten-year anniversary paper summarizes the roots, design thinking and the evolution of the Lively Kernel from the technical perspective over the past ten years [8].

The broader software development challenges that we faced in the early years were summarized in another paper that was provocatively called “Spaghetti Code for the 21st Century” [1, 9]. In that paper, we argued that web development had reintroduced many of the spaghetti code problems that had already largely been eliminated in the software industry some ten years earlier. We listed issues that plagued web application development at the time, reminiscing us of the fabled “spaghetti code wars” in the early 1970s. We argued that web development was effectively giving rebirth to many of the same issues that were identified two decades earlier as the main culprits for unreadable, unmaintainable code.

Since then, we have been involved in the development of various other projects related to web development. In this paper, a revisited version of an earlier conference paper [10], we reflect upon our twelve-year adventure in web development, focusing especially on our learnings on software development challenges associated with web-based software development. We will revisit various topics that we identified as central challenges in web development over ten years ago. Although things have generally been moving in a better direction, we argue that the “organic”, rather uncontrolled evolution of the Web and the dramatic increase in popularity of web-based software development in general have exacerbated the problems and the “impedance mismatch” between web development and software engineering [1, 11]. We will also present some interesting research opportunities and directions for the next ten years.

The structure of this paper is as follows. We start the paper with a review of the software engineering principles in the context of the Web, revisiting the

central challenges that we identified over ten years ago (Sect. 2). We then take a look at the state of web programming today, highlighting significant changes in web development since we started our journey many years ago (Sect. 3). In Sect. 4, we compare the state of the art in web development today to our earlier findings. In Sect. 5, we present some additional observations and technical challenges, followed by some reflections and forward looking predictions in Sect. 6. Finally, Sect. 7 concludes the paper.

2 Software Engineering Principles in the Context of Web Programming

This section provides a condensed summary of our “Spaghetti Code for the 21st Century” paper, published as a Sun Labs Technical Report in June 2007 [9] and as a conference paper (in somewhat shorter form) in 2008 [1]. The challenges identified in those publications serve as the backdrop for the evaluation and discussion later in this paper.

Back in 1968, Edsger Dijkstra started his crusade against “spaghetti code” [12]. Spaghetti code is a pejorative term for source code that has a complex and tangled control structure, especially one using many *gotos*, exceptions, threads, global variables, or other “unstructured” constructs. It is named such because program flow tends to look like a twisted and tangled bowl of spaghetti. The term is commonly used in negative sense to imply that a given piece of work is difficult to understand.

As underlined by the spaghetti code controversy, software engineering remained an undeveloped, unestablished practice until the late 1970s [13, 14]. Many important principles, such as *modularity*, *information hiding*, *separation of concerns* (especially the separation of *specification from implementation*), *manifest interfaces*, *reusability* and *portability* did not exist or were not adopted widely until they were introduced and instituted by Parnas, Clements, Corbató, Dahl, Guttag, Hoare, Morris, Liskov, Zilles and many others in the seminal articles in the 1970s and 1980s [15–30]. MacLennan has summarized many of these principles in his book that focuses on the principles of programming languages [31].

An important milestone in the codification of software engineering is the 1968 NATO Software Engineering Conference [32]. Besides introducing the idea of reusable software and software components [33], the attendees of the conference agreed that design concepts essential to maintainable systems are *modularity* (to isolate functional elements of the system), *specification* (of the interface as opposed to the implementation), and *generality* (required for extensibility). In that conference, the first formal definition of software engineering was also specified. As summarized by Bauer in [32], software engineering was then defined as the “*establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.*” Today, some fifty years later, that definition is still as valid as ever.

As long as the primary purpose of web development was the creation of web sites consisting of documents, pages and forms, there was little reason to apply established software engineering principles to web development. The web browser, with its original design dating back to 1990, was and still is well-suited to displaying documents and supporting simple navigation from page to page. However, over the years web pages have increasingly taken the form of desktop-style applications, with richer user interface and direct manipulation capabilities, as well as more advanced asynchronous communication between the clients and servers. The size and complexity of web applications and pages have also grown dramatically¹. This has increased the need to treat web development in the same fashion as software development. An *ad hoc*, document-oriented and tool-driven approach to web development – as was common especially in the early days of web development – is insufficient in this regard.

In many ways, web application development in its early days was reminiscent of software development in the 1970s before software engineering principles were defined and systematically applied to software development. The nascent field of *Web Engineering* has emerged in response to the need to introduce sound engineering principles to web development; the first international conferences in this area (such as the ICWE conferences) were arranged in the early 2000s.

In our “Spaghetti Code for the 21st Century” paper, we grouped the challenges in web application development into three main categories based on well-documented software engineering principles. These categories were: (1) *modularity and interfaces*; (2) *consistency, simplicity and elegance*; and (4) *reusability and portability*. Furthermore, identified two additional categories of important challenges related to (4) *usability* and (5) *development style*.

Below we provide a condensed summary of the key challenges that we identified back in 2007. These challenges will serve as a backdrop for the updated discussion in the rest of this paper. The abbreviations and numbers in parentheses below (e.g., SC#1, WI#2) will be used later in this paper to refer to the earlier identified issues.

– Modularity and Interface Issues

- *Separation of Concerns (SC#1)*: Declarative and procedural development style are mixed up.
- *Separation of Concerns (SC#2)*: User interface component placement, user interface style elements, event declarations and application logic are mixed up.
- *Separation of Concerns (SC#3)*: Dependence on tool support.
- *Well-Defined Interfaces (WI#1)*: No well-defined interfaces exist between the browser and other components, apart from the Document Object Model (DOM).
- *Well-Defined Interfaces (WI#2)*: Hard-coded references and other implementation details are used openly.

¹ In 2016, Wired Magazine reported that the size of the average web page had exceeded the size of the original (year 1993) Doom multiplayer 3D computer game (<https://www.wired.com/2016/04/average-webpage-now-size-original-doom/>).

- *Information Hiding (IH#1)*: The DOM tree is exposed and manipulated through side effects.
 - *Information Hiding (IH#2)*: Source code of applications is exposed.
 - *Information Hiding (IH#3)*: No privacy mechanisms available in JavaScript.
- **Consistency, Simplicity and Elegance Issues**
- *Consistency (C#1)*: There are several ways to perform the same functions.
 - *Consistency (C#2)*: Things should happen explicitly rather than through side effects.
 - *Simplicity and Elegance (SE#1)*: Web applications are unstructured and hard to read.
 - *Simplicity and Elegance (SE#2)*: Different types of technologies (e.g., HTML, JavaScript, CSS, XML) are mixed up.
- **Reusability and Portability Issues**
- *Reusability (R#1)*: Elements of reuse are scattered and mixed with the rest of the application.
 - *Reusability (R#2)*: Hard-coded references and other implementation details are exposed.
 - *Portability (P#1)*: There are still significant differences between browsers and browser versions.
 - *Portability (P#2)*: Portability of (developer) experience is poor.
- **Usability Issues**
- *Usability (U#1)*: The browser I/O model is poorly suited to desktop-style applications.
 - *Usability (U#2)*: The semantics of many browser features are unsuitable for applications.
- **Development Style Issues**
- *Development Style (DS#1)*: No transitive closure of program structures is available statically.
 - *Development Style (DS#2)*: There is no support for static verification or static type checking.

Each of the issues was discussed in detail in the original technical report [9] and conference paper [1]. For a detailed description of the issues listed above, refer to those papers. In general, our earlier studies pointed out an *impedance mismatch* between software engineering and web development – the former was already an established discipline with well-defined, mature methodologies and commonly understood engineering practices, while the latter was based on a combination of *ad hoc* practices and tools. Let us next examine the state of the art in web development some twelve years later.

3 State of Web Programming Twelve Years Later – General Observations

In this section we will take a look at the state of the art in web programming today, approximately twelve years later since our original analysis. We will begin

with a general overview of the changes that have occurred in the past several years. Later in the paper we will then reflect and map the present state in the industry to our original findings.

3.1 The Web and the Software as a Service (SaaS) Model Have Redefined Personal Computing

Today, the use of the Web as a software platform and the benefits of the Software as a Service model are widely understood [4,34]. For better or worse, the web browser has become the most commonly used desktop application; often the users no longer open any other applications on their PCs than just the browser. Effectively, for many average computer users today, the browser *is* the computer. A recent VisionMobile developer survey report strongly confirmed this observation, proposing the following key trends [6]:

- The browser has become the default interface for desktop applications.
- If the browser isn't used to run the desktop app, it is being used to distribute it.
- ChromeOS is gaining a foothold in Southern Asia.

Based on the points above, it is fair to say that the Web and the Software as a Service model have redefined the notion of personal computing. Although conventional desktop applications do still exist and are still widely used, desktop applications and their deployment model are now primarily web-based. Perhaps the most representative example of this ongoing paradigm shift is Microsoft's web-based Office 365 productivity suite (<https://www.office.com/>) that replaces Microsoft's earlier (native) Office suite – the most iconic and prevalent software product of the earlier PC era. This trend has also sparked the introduction of totally new computing device categories, such as Google's purely browser-based Chromebook personal computers (<https://www.google.com/chromebook/>) running the web-based ChromeOS operating system.

3.2 JavaScript Has Become a Very Popular Programming Language

Due to the central role of the web browser, JavaScript has become one of the most popular programming languages in the world, just as we anticipated ten years ago. While JavaScript language standardization work was stalled for many years, there is now major progress on the standards front. The ECMAScript 6 Specification was finally published in June 2015 [35], followed by ECMAScript 7 Specification a year later [36]. Although the suitability of the JavaScript language for large masses of software developers can still be debated, ECMAScript 6 – also known as ECMAScript 2015 – is actually a decent and expressive programming language, providing support for features such as modules, class declarations, lexical block scoping, iterators and generators, promises for asynchronous programming, and proper tail calls. Furthermore, libraries and tools have come to rescue for numerous other problematic characteristics. For instance, Flow (<https://flow.org/>) is a static type checker for JavaScript programs, which can also determine the completeness of applications.

3.3 Interactive, Visual Development on the Web Has Become Commonplace

From the viewpoint of the original Lively Kernel vision (see [7]), it is interesting to note that interactive, visual development for the Web has become commonplace. There are numerous interactive HTML5 programming environments such as *Cloud9* (<https://c9.io/>), *Codepen.io* (<http://codepen.io/>), *Dabblet* (<http://dabblet.com/>), *JSBin* (<https://jsbin.com/>), *JSFiddle* (<https://jsfiddle.net/>), and *Plunker* (<https://plnkr.co/>) that capture many of the original qualities of the Lively vision – such as the ability to perform software development entirely within the confines of the web browser. In the research front, CoRED has investigated the possibilities of collaborative coding [37].

In addition, there are *web curation systems* (see [38]) and JavaScript visualization libraries such as *Chart.js* (<http://www.chartjs.org/>), *Cola.js* (<https://github.com/tgdwyer/WebCola>), *D3* (<https://d3js.org/>), and *Vis.js* (<http://visjs.org/>) that provide rich, interactive, animated 2D and 3D visualizations for the Web, very much in the same fashion as we envisioned when we started the work on the Lively Kernel back in 2006. A central difference, though, is that these new libraries are intended primarily for data visualization rather than for general-purpose application development.

3.4 Web Browser Performance and JavaScript Performance Have Improved Dramatically

While the original versions of the Lively Kernel ran slowly, advances in web browsers and high-performance JavaScript engines soon changed the situation dramatically. The emergence of Google’s Chrome web browser and the V8 JavaScript engine – created at Google by some of our former colleagues from Sun Microsystems – kick-started web browser performance wars. Raw JavaScript execution speed increased roughly by three orders of magnitude between years 2006 and 2013, effectively repeating the same dramatic performance advances that had occurred with Java virtual machines ten years earlier when those VMs evolved from simple interpreter-based systems to using advanced adaptive just-in-time compilation techniques. Although improvements in the UI rendering area have been less dramatic, from the end user’s perspective today’s web browsers are easily 10–20 times faster than ten years ago [39]. This has made it possible to run serious applications in the web browser. (Sadly, this has also enabled much richer use of interactive advertisements on web sites.)

3.5 HTML, CSS and the DOM Turned Out to Be Much More Persistent Than Anticipated

The browser and JavaScript performance improvements – while definitely impressive – were not really unforeseen to us. We were convinced that the performance problems of the browser and JavaScript would ultimately get resolved.

However, what was unforeseen to us how “sticky” the original core technologies in web development – HTML, CSS and JavaScript – as well as the use of the Document Object Model (DOM) would be. Our assumption was that software developers would prefer having a more uniform, conventional set of imperative graphics APIs – supporting direct, programmatic object manipulation much in the same fashion as in conventional desktop operating systems – instead of using features that were originally designed for document layout rather than for programming.

Furthermore, when we gave presentations in web developers conferences in the late 2000s, reminding web developers of traditional software engineering principles such as modularity, separation of concerns and the general importance of keeping specifications and public interfaces separate from implementation details [19], web developers shrugged and noted that the use of HTML, CSS and JavaScript already gave them the necessary separation. Likewise, the ability to manipulate graphics by poking the global DOM tree from anywhere in the application was seen as a perfectly acceptable way of doing things rather than as something that would raise any serious concerns.

In recent years, things have gone in a better direction given the earlier mentioned modularity mechanisms that have been added to the ECMAScript language, increasing use of RESTful APIs, as well as upcoming support for *Web Components* (<https://www.w3.org/TR/#tr-Web.Components>). Web Components bring component-based software engineering principles to the World Wide Web, including the interoperability of higher-level HTML elements, encapsulation, information hiding and the general ability to create reusable, higher-level UI components that can be added flexibly to web applications.

3.6 Instant Worldwide Deployment and Dramatically Faster Release Cycles Have Become Commonplace

When the Lively Kernel project was started, the majority of software deployments at Sun Microsystems were still done in a conventional fashion by distributing physical CDs/DVDs or by making new binary installers available on the Web. New software releases occurred relatively infrequently, perhaps a few times per year for major software products such as the Java SDK. In contrast, web-based systems allow changes to be published pretty much instantly worldwide.

Since the Lively Kernel was one of the first systems to boldly enter such an instant deployment model, we had no support from tools and techniques that have later been introduced in the context of continuous deployment [40]; this has given rise to an entirely new development process around associated automation and tools. In hindsight, it is amazing how quickly the traditional deployment model was replaced by instant worldwide deployment enabled by the Software as a Service model. This has resulted in dramatically faster release cycles as well as in the rise of entirely new continuous development and deployment practices methodologies across the industry, including DevOps [41]. These topics are now so widely studied and documented that we do not need to dive more deeply into them in this paper. For details the reader is referred to [40, 42, 43].

4 Comparing Then and Now – Reflections on Software Engineering Principles

In our original Spaghetti paper, we divided the challenges in web-based software development into three main categories and two additional areas based on established software engineering principles. We will now reflect on the present state of the art in web programming in light of those five categories. Instead of revisiting each earlier identified issue in detail, we will provide an overview of the key changes in the past decade in tabular format, complemented with some discussion.

4.1 Revisiting the Modularity and Interface Issues

Originally, the most essential modularity issues that we identified were related to the mixture of procedural and declarative programming style, the mixing of HTML/CSS/JS code, and the common assumption that development tools would ultimately solve the problems (instead of somebody actually addressing the underlying core issues) [1]. We also lamented the lack of well-defined interfaces and inadequate information hiding capabilities, as well as the openly exposed nature of the DOM tree and the source code comprising web pages.

Today, most of the previously identified challenges are still present at the browser level. For instance, the global nature of the DOM tree has not changed much, and web development is generally still a mixture of procedural and declarative development styles. However, improvements in the JavaScript/ECMAScript language as well as the availability of richer, better-designed libraries and frameworks has changed web development significantly. While in the earlier days the number of JavaScript libraries was very limited, today there is an extremely rich library ecosystem available for web development, including the currently dominant Angular.js and React.js developer ecosystems.

Table 1 presents a condensed evaluation of the main changes related to the modularity and interface issues that we identified back in 2007 [9]. For each previously identified issue we provide a short evaluation and comments on whether the issue has improved or worsened over the years.

4.2 Revisiting the Consistency, Simplicity and Elegance Issues

The second big bucket of challenges in our original Spaghetti paper was related to consistency, simplicity and elegance issues. One of the most significant observations back then was that the standards-compatible web browser offered too many ways to perform the same functions. We also lamented the unstructured nature of web applications, as well as the general tendency for web developers to mix different types of technologies, paradigms and development styles.

Table 2 presents a condensed evaluation of the main changes related to the consistency, simplicity and elegance issues that we identified back in 2007 [9]. For each previously identified issue we provide a short evaluation and comments on whether the issue has improved or worsened over the years.

Table 1. Modularity and interface issues revisited.

Issue	Evaluation	Comments
SC#1	Improved	Today, developers rarely perform low-level DHTML programming anymore. Thus, accidental mixing of programming paradigms is less common. The programming paradigm (declarative vs. procedural) is determined largely by the choice of libraries/frameworks on top of the web browser
SC#2	Improved	Today, developers rarely use the low-level DHTML mechanisms for component placement or event declarations directly. Thus, accidental mixing of different types of declarations is less common. Today, these aspects are driven largely by the choice of libraries/frameworks on top of the browser
SC#3	Neutral	In many ways, dependence on tool and library support in web development has increased considerably over the years. Whether this is a good or bad trend is subject to debate and personal preferences. The availability of richer and more mature development frameworks has definitely alleviated many of the issues identified earlier
WD#1	Mostly neutral	The number of APIs provided by a standards-compatible web has increased over the years. However, the DOM is still the primary communication interface inside the browser
WD#2	Improved	Today, developers rarely perform low-level DHTML programming anymore. Thus, hard-coded references or other implementation details are not as exposed as earlier. The actual improvements are dependent on the choice of libraries/frameworks on top of the browser
IH#1	Slightly improved	The DOM is still an exposed, global data structure. Web Components ameliorate the issues by providing support for encapsulation, information hiding and the general ability to create encapsulated DOM elements. However, Web Components are not in widespread use yet
IH#2	Neutral	The source code of web applications is still as exposed as earlier. Obfuscation techniques are used commonly by developers to hide source code
IH#3	Improved	JavaScript/ECMAScript language improvements (e.g., modules and proper lexical block scoping) that have improved the situation and reduced the danger of accidental name clashes

Discussion. In the consistency, simplicity and elegance area, things have generally been moving in a better direction. That said, today's web application developers are faced with an even more overwhelming cornucopia of choices in

Table 2. Consistency, simplicity and elegance issues revisited.

Issue	Evaluation	Comments
C#1	Worsened	The number of programming models offered by a generic web browser has actually increased over the years. For instance, the introduction of the WebGL API has made it possible to perform web rendering using yet another built-in API. Also, the number of libraries and frameworks has increased dramatically over the years. Thus, developers are faced with even more choices and even more ways to perform the same functions
C#2	Improved	Today, developers rarely use the low-level DOM manipulation operations directly. This has reduced the use of programming styles that rely on side effects. The actual improvements in this area are dependent on the choice of libraries/frameworks on top of the web browser
SE#1	Improved	The availability of mature, higher-level libraries and frameworks has improved the overall quality and structure of web applications considerably. Again, the actual improvements in this area are driven mainly by the choice of libraries/frameworks
SE#2	Improved	Today, developers rarely perform low-level DHTML programming anymore. Thus, accidental mixing of different types of technologies and paradigms is less common. The actual improvements in this area are dependent on the choice of libraries/frameworks on top of the browser

almost all aspects of web development. A great example are the rendering mechanisms inside the web browser that we have studied recently in [44]. The basic observation in that article is that the generic, standards compatible web browser offers five overlapping rendering models: DHTML, Canvas API, WebGL, SVG and Web Components. The developers are confronted with various choices also in choosing communication models, e.g., whether to use Ajax [45], Comet [46], Server-Sent Events [47], WebSockets [48], WebRTC [49]), or Web Workers [50].

In the broader picture, the deficiencies of the web browser as a software platform have been tackled with an abundance of libraries. As of this writing, there are more than 1,300 officially listed JavaScript libraries in javascripting.com, with new ones being introduced nearly on a daily basis. Although many of the libraries are domain-specific, a lot of them are aimed squarely at solving the architectural limitations of the web browser, e.g., to provide a consistent set of manifest interfaces to perform all the programming tasks. Over the years, JavaScript libraries have evolved from mere convenience function libraries to full-fledged Model-View-Controller (MVC) frameworks providing extensive UI component sets, application state management, network communication and database interfaces, and so on.

4.3 Revisiting the Reusability and Portability Issues

The third bucket of challenges in our original Spaghetti paper was related to reusability and portability. In our earlier paper we criticized the generally unstructured nature of web applications that made it difficult to isolate and package components for reuse. We also lamented the incompatibilities between different browsers and browser versions that made it burdensome to write portable code that would work across different browsers and browser versions. Furthermore, we noted that the abundance of different libraries and frameworks reduced the overall portability of developer experience, since development guidelines and recommended practices for one library were typically different from other libraries. Table 3 presents a condensed evaluation of the main changes related to the reusability and portability issues that we identified back in 2007 [9].

Discussion. In the reusability and portability area, things have also been moving in a better direction, primarily because the richer development frameworks, the introduction of Web Components and new JavaScript language mechanisms have encouraged the developers to write considerably more structured code and components specifically intended for reuse.

However, just like ten years ago, a central problem in web application development is browser incompatibility. While browser compatibility has generally improved significantly over the years, the rapid pace of innovation and constant introduction of new features has kept browsers pacing each other as browser vendors have prioritized their implementation roadmaps differently. Over the years, there have also been business and legal reasons for some of the incompatibilities, such as the intellectual property rights issues in the media codec area. Because of incompatibilities, developers still often depend on compatibility bridge libraries such as Modernizr (<https://modernizr.com/>) that detect missing features in the underlying browser and fill in the gaps automatically.

4.4 Revisiting the Usability and User Experience Issues

The fourth bucket of challenges in our original Spaghetti paper was related to usability. In our earlier paper we focused the usability analysis on the impedance mismatch between traditional desktop applications and the document-oriented, page-oriented application model introduced by the web browser. We noted that in web applications user interaction was based primarily on pages and hyperlinks, as opposed to PC applications that supported modern (or at least modern back then) user interaction features such as direct manipulation, menu-oriented navigation, and a rich set of interactive graphical widgets.

Table 4 presents a condensed evaluation of the main changes related to the usability and user experience issues that we identified back in 2007 [9].

Discussion. Overall, the user experience of the web browser has not changed very much from the page-oriented *back-forward-reload* metaphor introduced by

Table 3. Reusability and portability issues revisited.

Issue	Evaluation	Comments
R#1	Improved	Today, developers rarely perform low-level DHTML programming anymore. Thus, a programming style that spreads elements of reuse in a spaghetti-like fashion is less common. Web Components make it possible write web UI components with proper encapsulation and information hiding. The actual improvements in this area are dependent on the choice of libraries/frameworks on top of the web browser (including the option to use Web Components)
R#2	Improved	Today, developers rarely perform low-level DHTML programming anymore. Thus, the use of hard-coded references is significantly less common. The actual improvements in this area are dependent on the choice of libraries/frameworks and/or Web Components
P#1	Improved	Browser compatibility has improved significantly over the years. For instance, the event handling capabilities of Microsoft browsers are now compatible with the other major web browsers. There are also much better compatibility test suites available nowadays. Then again, the rapid introduction of new browser features and APIs has a tendency to keep browsers somewhat incompatible with each other, as the browser vendors struggle to implement all the latest features
P#2	Neutral or somewhat worsened	Regarding the portability of developer experience, the abundance of web technologies has made things even more challenging for developers, as it is not necessarily very easy to migrate skills learned with one library ecosystem to another. For instance, the transition from Angular.js development to React.js can be demanding. However, given that the majority of today’s software developers grew up with web technologies, they are less burdened by patterns and conventions learned during the earlier desktop software era

the NCSA Mosaic browser in the early 1990s [51]. Ten years ago, the page-oriented interaction and navigation model of the web browser seemed like a throwback to an earlier era. We remarked that the interaction model of the web browser was reminiscent of the I/O model of the IBM 3270 series terminals of the 1970s – in both systems the entire display was updated in response to each successful user-initiated network request. We also lamented the poorly defined semantics of many of the browser functions and buttons. For instance, semantics of the ‘back’, ‘stop’ and ‘reload’ buttons were unclear when these features were used in desktop-style web applications.

Table 4. Usability issues revisited.

Issue	Evaluation	Comments
U#1	Neutral	The page-oriented user interaction model can still be considered suboptimal for desktop applications. The introduction of more advanced communication capabilities in the web browser, such as asynchronous HTTP requests [45] and Server-Sent Events [47], have alleviated issues considerably, though. More broadly, the web browser has become such a dominant environment for applications that our earlier observations are mostly irrelevant nowadays
U#2	Neutral	The semantics of many of the browser buttons and other features (e.g., the context menus that open up when right-clicking objects in a desktop browser) are still poorly suited to applications. However, the web browser has become such a dominant environment for applications that our earlier observations are mostly irrelevant nowadays

Even today, it still is not entirely clear to the user what will happen if the user presses the ‘back’, ‘stop’ or ‘reload’ button during a financial transaction initiated from a browser-based banking or stock trading application. To avoid potentially harmful (and expensive) interactions, some web applications explicitly disable many of the browser’s navigation buttons.

Interestingly, even though the observations that we presented earlier are still valid today, the usability issues have become mostly irrelevant because of the dominant role that the web browser and the Software as a Service (SaaS) model have today. Nowadays, the majority of computer users are so accustomed to the web browser and its user interface that they rarely miss the features or conventions of the earlier PC-era desktop applications. Browser-based interaction has simply become the norm also for desktop applications². For the average computer user, the web browser effectively is *the* application platform now.

It should be noted that the adoption of Single-Page Application (SPA) development style [52, 53] and its support in popular frameworks such as Angular.js [54] have improved the overall user experience considerably for those web sites that wish to behave more like classic desktop applications. Nevertheless, we still think that a lot of room remains in improving the overall usability of web applications. Those discussions are beyond the scope of this paper, however.

4.5 Revisiting the Development Style Issues

The fifth and last bucket of challenges in our original Spaghetti paper was related to development style. We pointed out that the field of web programming bears

² Even this paper was written using the online tool *Overleaf* instead of a conventional, PC-based word processing application such as Microsoft Word.

the imprint of the document-oriented – as opposed to application-oriented – roots of the Web. The programming capabilities of the Web have largely been an afterthought – designed originally for relatively simple scripting tasks. For instance, the JavaScript language was originally created by Brendan Eich in ten days in May 1995.

In our earlier paper the analysis of development style issues focused primarily on the differences that arise from the use of dynamic vs. static programming languages. Ten years ago, the software development landscape was dominated by statically compiled programming languages such as C, C++ and Java. The use of interpreted, dynamic programming languages such as Lisp, Scheme, Python or JavaScript was limited. Today, dynamic languages (especially JavaScript and Python) have a central role especially in client-side development, driven by the expectation and need to have much shorter release cycles and the ability to perform changes in near real time [55].

Table 5 presents a condensed evaluation of the main changes related to the development style issues that we identified back in 2007 [9].

Table 5. Development style issues revisited.

Issue	Evaluation	Comments
DS#1	Improved	An abundance of tools is available nowadays to ensure the completeness of web applications. For instance, the earlier mentioned Flow tool (https://flow.org/) is a static type checker for JavaScript programs that can also determine the completeness of applications. Tools such as Webpack module bundler can also help in packaging applications and make sure they contain all the necessary components
DS#2	Improved	An abundance of tools is available nowadays to support static verification and static type checking of web applications. There are also JavaScript language extensions and variants such as TypeScript that make optional type checking available with the latest ECMAScript features (https://www.typescriptlang.org/)

5 Comparing Then and Now – Additional Observations

In our WEBIST conference paper published in 2017, we listed a number of additional topic areas and web application development issues that were not covered by our original Spaghetti evaluation framework presented over ten years earlier. Below we summarize these additional observations and challenges. Some of these topics arise from the security limitations of the web browser, while other topics arise from the dramatically faster development cycles that have become commonplace in the past 5–10 years.

Limited Access to Local Resources or Host Platform Capabilities. Web documents and applications are run in a sandbox that places significant restrictions on the resources and host platform capabilities that the web browser can access. For instance, access to local files on the machine in which the web browser is being run is not allowed, apart from reading and writing cookies and using the `localStorage` mechanism. While these security restrictions prevent malicious access, they make it difficult to build web applications that utilize local resources or host platform capabilities. Consequently, the functionality that can be offered by web applications is still significantly more limited than that of native applications. In this area things have not changed very much in the past ten years.

Mobile Computing is Still Dominated by Apps – For Now. During the original development of the Lively Kernel system in 2006–2008, we were aiming at making the system run well also on mobile web browsers. Although the feasibility of running the system on mobile devices was demonstrated [56], in practice mobile devices and browsers were still so slow those days that no serious mobile Lively applications could be built. Furthermore, the considerably smaller screen sizes, different input modalities and limited mobile OS API access made it difficult to develop and run applications on mobile devices.

The technical reasons for the desktop and mobile app divergence are well understood nowadays [57, 58]. A major contributor to the divergence is the limited access to the underlying platform capabilities mentioned above. One approach for tackling the shortcomings of the Web as a mobile platform is to use cross-platform or hybrid app designs [3, 59]. In the late 2000s, so called *Rich Internet Application* (RIA) platforms such as Adobe AIR, Apache Cordova [60] (formerly PhoneGap) and Microsoft Silverlight [61] were very popular. RIA systems were an attempt to bring alternative programming languages and libraries to the Web in the form of browser plug-in components that each provided a complete, more efficient platform runtime (see [3]). However, just as it was predicted in [62], the RIA phenomenon turned out to be rather short-lived.

More broadly, it is interesting to note that in the past ten years desktop computing and mobile computing have evolved in entirely different directions. While personal computers are now driven mostly by the Software as a Service model, mobile devices are still dominated by native or hybrid apps. This divergence is unlikely to continue indefinitely. There are already indications that desktop and mobile operating systems will ultimately converge. For instance, Microsoft’s latest Windows 10 Mobile operating system represents an attempt to unify Windows application platform across multiple device classes.

Fine-grained Security Model is Still Missing. Compared to traditional desktop applications, web applications can still be viewed as second-class citizens that are at the mercy of the classic, one size fits all sandbox security model of the web browser. This means that decisions about security are determined primarily by the site (origin) from which the application is loaded, and not by the specific needs of the application itself. The situation is further complicated by

opportunistic designs and mashware paradigm, where applications are composed out of data and code available from various web sites [11, 63].

Testing of Web Applications is Still Challenging. Related to testing, web applications are generally so dynamic that it is impossible to know statically – ahead of application execution – if all the structures that the program depends on will be available at runtime. While web browsers are designed to be error-tolerant and will ignore incomplete or missing elements, in some cases the absence of elements can lead to fatal runtime problems that are impossible to detect before execution. Furthermore, with scripting languages such as JavaScript applications can even modify themselves on the fly, and there is no way to statically detect the possible errors resulting from such modifications. Consequently, web applications require significantly more testing to make sure that all the possible application behaviors and paths of execution are covered.

Forgiveness and Error-Tolerance. The web browser and the JavaScript virtual machine have been designed to be extremely permissive and error-tolerant. As a general principle, errors are not reported unless absolutely necessary. For instance, spelling errors in JavaScript variable names implicitly result in the creation of a new variable with the misspelled name. Likewise, minor accidental syntax errors, such as using square brackets “[]” instead of parentheses “()”, e.g., in string indexing operation ‘‘`String.chatAt()`’’, will go unreported and can lead to problems that are very difficult to trace. While such permissiveness enables the successful execution of source code that contains spelling errors, this usually results in other, significantly more difficult errors later in the execution. When an error is finally reported, the actual problems hides elsewhere in the program. Such problems multiply when creating web-based mashups that combine code from multiple sources and different authors [11, 64].

Challenges arising from the forgiving, permissive nature of the web browser and JavaScript are tackled primarily by tools. However, declarative applications are not easy to test with present-day tools; they rely on, e.g., test coverage that has little meaning in a declarative setup. Furthermore, coverage testing has very little meaning if the application relies extensively on external libraries and modules. For instance, in a typical Node.js application today, the amount of actual application code is often marginal compared to the thousands of NPM modules that the application uses. In such an environment, actual application code may only consist of a few hundred or thousand lines, while the NPM modules used by the application contain millions of lines of code from external sources.

Fashion-Driven Development. Over the past years there has been a notable trend in the library area towards fashion-driven development. By this we refer to the developers’ tendency to surf on the wave of newest and most dominant “alpha” frameworks. For instance, the once hugely popular *Prototype.js* and *jQuery.js* libraries are nowadays mostly forgotten, replaced by *Knockout.js* and *Backbone.js* in 2012. Back in 2014, *Angular.js* was by far the most dominant alpha framework, while in 2016–2017 it is the *React.js + Redux.js* ecosystem that is capturing the majority of developer attention. As witnessed by the some-

what unfortunate recent evolution of the Angular ecosystem, the alpha frameworks have a tendency to evolve very quickly once they get developers' attention, leading into compatibility issues. To make the matters worse, once the next fashionable major framework emerges and hordes of developers start jumping ship onto the new one, it becomes questionable to what extent one can build long-lasting business-critical applications and services, e.g., for the medical industry in which products must commonly have a minimum lifetime of twenty years. With the present pace of upgrades, the browser and the web server as the runtime environment would be almost completely replaced by patches, upgrades, and updates; similarly, most of the libraries would be replaced several times by newer ones.

Opportunistic Design and Cargo Cult Programming. In web development there has historically been a strong tradition of *mashup-based development* [11,64]: searching, selecting, pickling, mashing up and gluing together disparate libraries and pieces of software [63]. Often such development has the characteristics of *cargo cult programming*³: ritually including code and program structures that serve no real purpose or that the programmer has chosen to include because hundreds of other developers have done so – without really understanding why. While this approach can save a lot of work and open up interesting opportunities for large-scale code reuse [64,65], this approach does not foster development of reliable, long-lasting applications, because even the smallest changes in the constituent components or subsystems – each of which evolves separately and independently – can break applications [66]. In Node.js development, opportunistic design is especially common, as the developers often include numerous NPM modules for convenience, or simply because many their colleagues or other developers have done so.

6 Reflections and the Road Forward

In striking contrast with the situation twelve years ago, there is now an incredible amount of innovation in the web development area. End user software has largely migrated to the Web, JavaScript has become one of the most popular programming languages in the world, and new libraries and tools have become available almost on a weekly (if not daily) basis (see, e.g., <http://www.javascripting.com/>). The rapid pace of innovation and rather uncontrolled, organic evolution of the Web have resulted in a situation in which there are numerous ways to build applications on the Web – many more than most people realize, and also arguably more than are really needed. This has put the developers in a complex position in which it is difficult to choose technologies that would be guaranteed to still be around and supported ten years from now.

While the overall complexity of the web application development scene has increased, there have been improvements in nearly all the problem areas that we identified earlier. For instance, as already mentioned, there has finally

³ https://en.wikipedia.org/wiki/Cargo_cult_programming.

been tremendous progress in ECMAScript (JavaScript) language standardization [35, 36]. Furthermore, newer browsers – in particular Microsoft’s Edge browser⁴ that has replaced Internet Explorer – are significantly more compatible with each other than dominant browsers ten years ago. We are confident that similar compatibility improvements will find eventually their way also to mobile web browsers that still have more significant feature deviations today (there is a good overview available at <http://mobilehtml5.org/>).

In the same vein, Web Components offer hope that well-known (but hitherto missing) software engineering principles and practices will eventually find their way into the web browser, including modularity and the ability to create higher-level, general-purpose UI components that can be flexibly added to web applications. Web components are still the “dark horse” in web development – they are little known to most developers, and it is difficult to place betting odds on their eventual success. Web components cater to nearly any imaginable use case but they are especially well-suited to the development of full-fledged web applications that require an extensible set of GUI widgets.

Looking forward, we predict that the current transition towards the Internet of Things (IoT) and the Web of Things (WoT) will drive the industry towards systems that have much better support for interactive development and programming. We are moving to the *Programmable World Era* in which literally all everyday objects will be connected to the Internet and will have enough computing, storage and networking capabilities to host a dynamic programming environment, thus turning everyday objects remotely programmable [67, 68]. Such a dynamic programming vision is actually very close to one of our central goals when we started the Lively Kernel system development back at Sun Microsystems Labs in 2006 [8].

For better or worse, everyday objects around us will have more computing power, storage capacity and network bandwidth than computers that were used for entire computing departments in the 1970s and 1980s. The availability and presence of such capabilities will open up tremendous possibilities for entirely new types of applications and services. Many of the platforms under development for the IoT domain leverage Node.js, which effectively means that JavaScript may well become the *de facto* programming language for IoT applications as well.

The Internet of Things area offers a natural playground for dynamic programming capabilities provided by live object systems such as the Lively Kernel. To this end, we plan to harness and leverage the Lively environment as a web-based graphical end-user programming environment for IoT devices, with the goal to realize the broader Programmable World vision by implementing the same kind of direct manipulation capabilities that demonstrated earlier. The key difference is that rather than just making the World Wide Web more lively, we now aim at making the entire world around us programmable in an effortless and lively fashion (“Lively Things”) [8].

⁴ <https://www.microsoft.com/en-us/windows/microsoft-edge>.

7 Conclusions

The World Wide Web is the most powerful medium for information sharing in the history of humankind. Somewhat accidentally, the success of the Web has turned the web browser also into a dominant platform for end-user software. Today, the Software as a Service (SaaS) model is prevalent on desktop computers, while traditional installed applications still maintain a stronghold in the mobile application area.

Over ten years ago, we published a number of papers on the emergence of the Web as a software platform. We noted that the field of web programming bears the imprint of the document-oriented – as opposed to application-oriented – roots of the Web. We pointed out that the programming capabilities of the Web have largely been an afterthought – designed originally by non-programmers for relatively simple scripting tasks. We examined the state of the art in web software development in light of established software engineering principles, and enumerated issues that plagued web application development at the time. Those issues reminisced us of the fabled “spaghetti code wars” in the early 1970s.

In this paper, we have revisited our earlier findings and examined the state of the art in web software development today based on our experiences and learnings from various web development projects in the past twelve years. In almost all areas and issues that we identified a decade earlier, things have generally been moving in a better direction. However, at the same time the overall complexity of the web development landscape has increased considerably, reflecting the vast amount of innovation and interest in this space.

Furthermore, while there has recently been tremendous progress in JavaScript language evolution and in improving JavaScript performance, the majority of innovation has occurred in higher layers parts of the stack (e.g., in developing more comprehensive and powerful libraries and frameworks), leaving some of the core issues – such as the overall complexity of the web browser – still unaddressed.

Looking forward, we believe that interactive, web-based software development capabilities will become even more important in the future as the industry moves towards the *Programmable World Era* in which everyday objects around us will become connected and programmable.

References

1. Mikkonen, T., Taivalsaari, A.: Web Applications - Spaghetti Code for the 21st Century. In: Proceedings of the International Conference on Software Engineering Research, Management and Applications (SERA 2008, Prague, Czech Republic, 20–22 August 2008), pp. 319–328. IEEE Computer Society (2008)
2. Taivalsaari, A., Mikkonen, T., Ingalls, D., Palacz, K.: Web Browser as an application platform. In: 34th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2008, Parma, Italy, 3–5 September 2008), pp. 293–302. IEEE Computer Society (2008)

3. Casteleyn, S., Garrigós, I., Mazón, J.N.: Ten years of rich internet applications: a systematic mapping study, and beyond. *ACM Trans. Web* **8**(3), 18:1–18:46 (2014)
4. Turner, M., Budgen, D., Brereton, P.: Turning software into a service. *Computer* **36**(10), 38–44 (2003)
5. Petsas, T., Papadogiannakis, A., Polychronakis, M., Markatos, E.P., Karagiannis, T.: Rise of the planet of the apps: a systematic study of the mobile app ecosystem. In: *Proceedings of the 2013 Internet Measurement Conference*, pp. 277–290. ACM (2013)
6. VisionMobile: Cloud and Desktop Developer Landscape (2016). <http://www.visionmobile.com/product/cloud-and-desktop-developer-landscape/>. Accessed 5 Mar 2016
7. Taivalsaari, A., Mikkonen, T., Ingalls, D., Palacz, K.: Web Browser as an Application Platform: The Lively Kernel Experience. Technical report, TR-2008-175, Sun Microsystems Laboratories (2008)
8. Ingalls, D., Felgentreff, T., Hirschfeld, R., Krahn, R., Lincke, J., Röder, M., Taivalsaari, A., Mikkonen, T.: A world of active objects for work and play: the first ten years of lively. In: *Proceedings of SPLASH 2016 Onward! Track* (Amsterdam, The Netherlands, 30 October–4 November 2016), pp. 238–249 (2016)
9. Mikkonen, T., Taivalsaari, A.: Web Applications: Spaghetti Code for the 21st Century. Technical report TR-2007-166, Sun Microsystems Labs, June 2007
10. Taivalsaari, A., Mikkonen, T.: The web as a software platform: ten years later. In: *WEBIST 2017*, Porto, Portugal (2017)
11. Mikkonen, T., Taivalsaari, A.: The Mashware challenge: bridging the gap between web development and software engineering. In: *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, pp. 245–250. ACM (2010)
12. Dijkstra, E.W.: Letters to the editor: go to statement considered harmful. *Commun. ACM* **11**(3), 147–148 (1968)
13. Dijkstra, E.W.: Programming: from craft to scientific discipline. In: *International Computing Symposium*, pp. 23–30 (1977)
14. Hoare, C.: Programming: sorcery or science? *IEEE Softw.* **1**(2), 5 (1984)
15. Corbato, F.: Sensitive Issues in the Design of Multi-Use Systems. Technical report, DTIC Document (1968)
16. Parnas, D.L.: Information Distribution Aspects of Design Methodology (1971)
17. Dahl, O.J., Dijkstra, E.W., Hoare, C.A.R.: *Structured Programming*. Academic Press Ltd., London (1972)
18. Parnas, D.L.: A technique for software module specification with examples. *Commun. ACM* **15**(5), 330–336 (1972)
19. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. *Commun. ACM* **15**(12), 1053–1058 (1972)
20. Parnas, D.L.: On the design and development of program families. *IEEE Trans. Softw. Eng.* **1**, 1–9 (1976)
21. Parnas, D.L.: Designing software for ease of extension and contraction. In: *Proceedings of the 3rd International Conference on Software Engineering*, pp. 264–277. IEEE Press (1978)
22. Parnas, D.L., Clements, P.C., Weiss, D.M.: Enhancing reusability with information hiding. *Tutorial Softw. Reusability*, 83–90 (1983)
23. Parnas, D.L., Clements, P.C.: A rational design process: how and why to fake it. *IEEE Trans. Softw. Eng.* **2**, 251–257 (1986)
24. Morris Jr., J.H.: Protection in programming languages. *Commun. ACM* **16**(1), 15–21 (1973)

25. Morris Jr., J.H.: Types are not sets. In: Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, pp. 120–124. ACM (1973)
26. Liskov, B., Zilles, S.: Programming with abstract data types. In: ACM SIGPLAN Notices, vol. 9, pp. 50–59. ACM (1974)
27. Liskov, B., Zilles, S.: Specification techniques for data abstractions. In: ACM SIGPLAN Notices, vol. 10, pp. 72–87. ACM (1975)
28. Guttag, J.: Abstract data types and the development of data structures. *Commun. ACM* **20**(6), 396–404 (1977)
29. Zilles, S.N.: Procedural encapsulation: a linguistic protection technique. In: ACM SIGPLAN Notices, vol. 8, pp. 142–146. ACM (1973)
30. Corbató, F.J.: On building systems that will fail. In: ACM Turing Award Lectures 1990. ACM (2007)
31. MacLennan, B.J.: Principles of Programming Languages: Design, Evaluation, and Implementation. Oxford University Press, New York (1999)
32. Naur, P., Randell, B.: Software Engineering: Report of a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7–11 October 1968, Brussels, Scientific Affairs Division, NATO (1969)
33. McIlroy, M.D., Buxton, J., Naur, P., Randell, B.: Mass-produced software components. In: Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany, pp. 88–98 (1968)
34. Bouzid, A., Rennyson, D.: The Art of SaaS: A Primer on the Fundamentals of Building and Running a Successful SaaS Business. Xlibris (2015)
35. ECMAInternational: ECMAScript 2015 Language Specification, Standard ECMA-262, 6th Edn., June 2015. <http://www.ecma-international.org/ecma-262/6.0/>. Accessed 22 Feb 2017
36. ECMAInternational: ECMAScript 2016 Language Specification, Standard ECMA-262, 7th Edn., June 2016. <http://www.ecma-international.org/ecma-262/7.0/>. Accessed 22 Feb 2017
37. Lautamäki, J., Nieminen, A., Koskinen, J., Aho, T., Mikkonen, T., Englund, M.: CoRED: browser-based collaborative real-time editor for Java web applications. In: Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, pp. 1307–1316. ACM (2012)
38. Lupfer, N., Kerne, A., Webb, A.M., Linder, R.: Patterns of free-form curation: visual thinking with web content. In: Proceedings of the 2016 ACM on Multimedia Conference (MM 2016, Amsterdam, The Netherlands, 15–19 October 2016), pp. 12–21 (2016)
39. Wagner, J.L.: Web Performance in Action: Building Fast Web Pages. Manning (2016)
40. Leppänen, M., Mäkinen, S., Pagels, M., Eloranta, V.P., Itkonen, J., Mäntylä, M.V., Männistö, T.: The highways and country roads to continuous deployment. *IEEE Softw.* **32**(2), 64–72 (2015)
41. Debois, P.: DevOps: a software revolution in the making. *J. Inf. Technol. Manag.* **24**(8), 3–39 (2011)
42. Olsson, H.H., Alahyari, H., Bosch, J.: Climbing the “Stairway to Heaven” - a multiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In: 2012 38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 392–399. IEEE (2012)
43. Fitzgerald, B., Stol, K.J.: Continuous software engineering: a roadmap and agenda. *J. Syst. Softw.* **123**, 176–189 (2017)

44. Taivalsaari, A., Mikkonen, T., Pautasso, C., Systä, K.: Comparing the built-in application architecture models in the web browser. In: 2017 IEEE International Conference on Software Architecture (ICSA), pp. 51–54. IEEE (2017)
45. Garrett, J.J.: Ajax: A New Approach to Web Applications, 18 February 2005. <http://adaptivepath.org/ideas/ajax-new-approach-web-applications/>
46. Crane, D., McCarthy, P.: What Are Comet and Reverse Ajax? Springer (2009)
47. Hickson, I.: Server-Sent Events. W3C Recommendation 03 February 2015 (2015). <http://www.w3.org/TR/eventsource/>
48. Pimentel, V., Nickerson, B.G.: Communicating and displaying real-time data with websocket. *IEEE Int. Comput.* **16**(4), 45–53 (2012)
49. Bergkvist, A., Burnett, D.C., Jennings, C., Narayanan, A.: WebRTC 1.0: Real-time Communication Between Browsers. Working draft, W3C (2012)
50. W3C: W3C Schools - HTML Web Workers Example. http://www.w3schools.com/html/html5_webworkers.asp
51. Darken, R.: Breaking the mosaic mold. *IEEE Int. Comput.* **2**(3), 97 (1998)
52. Mesbah, A., Van Deursen, A.: Migrating multi-page web applications to single-page Ajax interfaces. In: 11th European Conference on Software Maintenance and Reengineering CSMR 2007, pp. 181–190. IEEE (2007)
53. Mikowski, M.S., Powell, J.C.: Single Page Web Applications: JavaScript End-to-End. Manning, Shelter Island (2013)
54. Jadhav, M.A., Sawant, B.R., Deshmukh, A.: Single page application using AngularJS. *Int. J. Comput. Sci. Inf. Technol.* **6**(3), 2876–2879 (2015)
55. Poulson, L.D.: Developers shift to dynamic programming languages. *IEEE Comput.* **40**(2), 12–15 (2007)
56. Mikkonen, T., Taivalsaari, A.: Creating a mobile web application platform: the lively kernel experiences. In: Proceedings of the 24th ACM Symposium on Applied Computing (SAC 2009), Proceedings, vol. 3, pp. 177–184 (2009)
57. Charland, A., Leroux, B.: Mobile application development: web vs. native. *Commun. ACM* **54**(5), 49–53 (2011)
58. Joorabchi, M.E., Mesbah, A., Kruchten, P.: Real challenges in mobile app development. In: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 15–24. IEEE (2013)
59. Dalmasso, I., Datta, S.K., Bonnet, C., Nikaein, N.: Survey, comparison and evaluation of cross platform mobile application development tools. In: 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 323–328. IEEE (2013)
60. Wargo, J.M.: Apache Cordova 4 Programming. Pearson Education (2015)
61. Moroney, L.: Microsoft Silverlight 4 Step by Step. Microsoft Press, Redmond (2010)
62. Taivalsaari, A., Mikkonen, T.: The web as an application platform: the saga continues. In: 37th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2011, Oulu, Finland, 30 August–2 September 2011), pp. 170–174. IEEE Computer Society (2011)
63. Hartmann, B., Doorley, S., Klemmer, S.R.: Hacking, mashing, gluing: understanding opportunistic design. *IEEE Pervasive Comput.* **7**(3), 46–54 (2008)
64. Taivalsaari, A., Mikkonen, T.: Mashups and modularity: towards secure and reusable web applications. In: 2008 23rd IEEE/ACM International Conference on Automated Software Engineering-Workshops, ASE Workshops 2008, pp. 25–33. IEEE (2008)
65. Salminen, A., Mikkonen, T.: Mashups: software ecosystems for the web era. In: IWSECO@ICSOB (International Conference on Software Business), pp. 18–32 (2012)

66. Salminen, A., Mikkonen, T., Nyrhinen, F., Taivalsaari, A.: Developing client-side mashups: experiences, guidelines and the road ahead. In: Proceedings of 14th International Academic MindTrek Conference: Envisioning Future Media Environments, pp. 161–168. ACM (2010)
67. Wasik, B.: In the Programmable World, All Our Objects Will Act as One. *Wired*, p. 462, May 2013
68. Taivalsaari, A., Mikkonen, T.: Roadmap to the programmable world: software challenges in the IoT era. *IEEE Softw.* **34**(1), 72–80 (2017)



Web Platform Assessment Tools: An Experimental Evaluation

Solange Paz¹ and Jorge Bernardino^{1,2(✉)}

¹ Polytechnic of Coimbra, ISEC,
Rua Pedro Nunes, Quinta da Nora, 3030-190 Coimbra, Portugal
a21230164@alunos.isec.pt, jorge@isec.pt

² Centre of Informatics and Systems, University of Coimbra,
Pinhal de Marrocos, 3030-290 Coimbra, Portugal

Abstract. Web search engines are used daily to find information, helping the user to surf the web. Web searching is the most popular online activity and although search engines regularly use updated indexes to run quickly and efficiently, they sometimes fail to keep the user on their page for a long time. As such, it is important to have the lowest delay in response time. Therefore, it is essential to understand what load is supported by each search engine by conducting load testing. These tests have the objective of optimizing the performance of the application being tested, thus verifying the maximum amount of data that is processed. In this paper we conduct a comparative analysis of the four most popular web platform assessment tools, Apache JMeter, Apache Flood, The Grinder and Gatling. As important as the search engine response time is the accuracy of returned results, that is, the amount of correct links related to what was searched for. For that reason, the accuracy of results returned by web search engines are also evaluated. In the experimental evaluation we use two tools: Apache jMeter and The Grinder, to compare with the web search engines: Google, Bing, Ask and Aol Search.

Keywords: Web assessment tools · Load testing · Performance tests
Results accuracy

1 Introduction

Web searching is the most popular online activity and web search engines are used daily helping the user to surf the web. People around the world access web search engines frequently to obtain links for the information they are looking for. The speed in returning a response to the user as well as the accuracy of returned results are very important, because although there are almost imperceptible differences in response time, users are aware of these faults and could get frustrated. In the event of a high response time from a web search engine, users tend to abandon their search. On the other hand, a faster search engine creates a better usability experience for the user. For this reason, it is important that the response time of a search engine is the lowest possible. To understand the maximum load supported by a search engine it is necessary to conduct performance tests, which focus exclusively on efficiency and reliability.

Thus, the performance test evaluates the behaviour of an application when it is submitted to a workload, that is, when it has several users interacting simultaneously with the application. The results of these tests reveal application sites where resources are being wasted or used inefficiently.

The development process of software includes a variety of activities where the probability of occurring errors is enormous. Therefore, software testing is critical to ensure the quality of product functionality, and as a final revision of its specification, design and code.

When performing tests during software development value is added to the product, as a test conducted correctly uncovers bugs that must be corrected before release to improve quality and reliability of a system.

Despite the great importance of these tests, sometimes they aren't executed for the reason that testing is a costly activity within development.

Depending on what system is being developed testing can be responsible for more than 50% of the costs [19]. Load tests are normally performed to identify the behaviour of a system subjected to a specific expected load, which can be a number of simultaneous expected users, the number of transactions per hour or a number of transactions made on system currently in test. These type of tests are ideal to verify if the application, server or database being tested remains stable during its usual workload. Load tests help to identify the maximum capacity of an application as well as any impediments that can interfere with its operation in terms of capacity.

There are a variety of testing tools in the market with different features and functionalities. The main purpose of all these tools is to simulate users accessing a particular application and later record the response time of the same, providing in most cases several formats of visualization of the response time. In this work we compare four of the most popular tools [24]: Apache JMeter; Apache Flood; The Grinder and Gatling. The tools are compared in terms of functionality, usability and performance. This comparison helps the selection of the best tool and it promotes the use of software testing tools.

Nowadays the content of a website is important as well as the speed at which it responds. Companies focus on improving the capability of a website's response to avoid losing users. To conduct a realistic evaluation of the tools, four popular web search engines are tested in terms of performance and accuracy of returned results: Google, Bing, Ask, and Aol Search. The experimental evaluation consisted in simulating the access of several users to the search engines, with searches with different number of words. Two test tools were used: Apache jMeter and The Grinder to calculate the elapsed time of the results.

As important as the search engine response time is the accuracy of returned results, that is, the amount of correct links related to what was searched for. For that reason, the accuracy of results returned by web search engines are also evaluated. The accuracy assessment involved the search of five questions in each of the search engines. We have registered the total number of links with correct information according to the search and selected the search engine that returns the highest number of correct links.

This paper is an extended version of the article titled "Comparative Analysis of Web Platform Assessment Tools" presented at the 13th Conference on Web Information Systems and Technologies [18]. It improves and complements the former in the

following aspects: the performance evaluation tests used two tools, Apache jMeter and The Grinder, and not only one; in the experimental evaluation we used two, three and four search words; and Google, Bing, Ask, and Aol Search are compared in terms of the accuracy of the results returned.

The rest of this paper is structured as follows. Section 2 presents a literature review on loading testing and Sect. 3 describes the various types of performance tests. Section 4 describes the four testing tools, and Sect. 5 the qualitative and quantitative analysis of these tools. Section 6 presents the performance tests performed on each web search engine. Section 7 presents the experimental evaluation of the accuracy of each web search engine. Finally, Sect. 8 states the conclusions and proposes some future work.

2 Related Work

Web applications are ubiquitous and need to deal with a large number of users. Due to their exposure to end users, especially customers, web applications have to be fast and reliable, as well as up-to-date. However, delays during the usage of the Internet are common and have been the focus of interest in different studies [12, 17].

Load testing is thus an important practice for making sure a web site meets those demands and for optimizing its different components [9].

The goal of a load test is to uncover functional and performance problems under load. Functional problems are often bugs which do not emerge during the functional testing process. Deadlocks and memory management bugs are examples of functional problems under load. Performance problems often refer to issues like high response time or low throughput under load. The first conference about testing software was organized in 1972, at Chapel Hill, where the works presented at the conference defended that performing tests is not the same as programming [15].

The existing load testing research mainly focuses on automatic generation of load test suites [1–3, 11, 14, 25]. There is limited work, which proposes the systematic analysis of the results of a load test to uncover potential problems. Unfortunately, looking for problems in a load test is a timeconsuming and difficult task. The work [27] flags possible functional problems by mining the execution logs of a load test to uncover dominant execution patterns and to automatically flag functional deviations from this pattern within a test.

In [28] the authors introduce an approach that automatically flags possible performance problems in a load test. They cannot derive the dominant performance behavior from just one load test, since the load is not constant. A typical workload usually consists of periods simulating peak usage and periods simulating off-hours usage. The same workload is usually applied across load tests, so that the results of prior load tests are used as an informal baseline and compared against the current run. If the current run has scenarios which follow a different response time distribution than the baseline, this run is probably troublesome and worth investigating.

The authors of [7] introduced a new integrated automation structure by Selenium and Jmeter. This structure shares the test data and steps, which is useful for switching in several kinds of tests for web applications. With the use of this software structure one

can improve extensibility and reuse of the tests, as well as the product quality. This work describes how to design the tests automation based in web details.

In [26] the authors proposed a usage model and a load model to simulate user behaviors which help generate a realistic load to the web application load test. They implemented a tool known as “Load Testing Automation Framework” for web applications load test. The tool is based in the two models mentioned above.

There are not many scientific works dedicated to the comparison of evaluation tools of web platforms. However, in [16] it is used four testing tools: Apache JMeter, HP LoadRunner, WebLOAD and The Grinder, with the objective of comparing these tools and identify which one is the most efficient. In the comparison the parameters used are cost, the load limit and the ease of use. The best tool of the comparison was jMeter, since it’s free, has a huge ability to simulate load and its interface is easy to use.

In [22] are described three open source tools (jMeter, soapUI, and Storm) and compared them in terms of functionalities, usability, performance and software requirements. This work concludes that jMeter is the tool that takes longer to respond to the tests, compared to the other two tools.

Khan [21] compares the Apache jMeter and HP LoadRunner tools in terms of performance and concludes that the best one is Apache jMeter. Selecting this tool as the best derived from the ease of installation, and ease of learning how to use.

Unlike previous papers, besides a comparison between four web platforms evaluation tools: Apache jMeter, Apache Flood, The Grinder and Gatling, in our work it is also performed an evaluation of four search engines: Google, Bing, Ask and Aol search relatively to its performance and the accuracy of results.

3 Performance Testing Types

Performance testing is important to analyze and monitor the performance of web applications. Performance testing verify the response time of an application, thus determining its scalability and reliability. This is also used to identify the critical bottleneck of a system, or gather other information as the hardware needed for the operation of the application.

Before going to market, the software system must be tested against speed, stability and scalability under a variety of load conditions. If the system is to be sold without performance testing, it can cause problems as the system slows down when it is accessed by multiple users simultaneously, which affects directly the reputation of the company and the expected sales goal.

There are several performance tests used to measure the performance of a system when it is subjected to a certain workload [6]. Table 1 describes the most common types of performance tests. According to the information in Table 1, performance tests are divided in six types: Load testing; Stress testing; Volume testing; Endurance testing; Spike testing e Scalability testing. All these kind of test aim at evaluating the performance of a certain system to test. For that end, loads are applied in the system in different ways.

Table 1. Performance testing types.

Type	Description
Load testing	Load testing refers to placing a load on the system and analyze its performance
Stress testing	Stress testing refers to a large number of inputs, making numerous, concurrent attempts to access a single web site
Volume testing	Refers to testing with a certain amount of load volume
Endurance testing	Refers to placing a load on a system over a certain time period and check its performance
Spike testing	Refers to the use of a sudden increase of load and analyze if the system behavior degrades or cope with the load changes
Scalability testing	Refers to the system capacity working under the expected load

4 Evaluated Tools

There are several tools available on the market, some of them free, while others are paid. From all the existing tools, four were selected which are considered the most popular and for its ability to measure web applications performance and its proficiency after a load test [25]. We will evaluate the following tools:

- Apache JMeter (<http://jmeter.apache.org/>)
- Apache Flood (<https://httpd.apache.org/test/flood/>)
- The Grinder (<http://grinder.sourceforge.net/>)
- Gatling (<http://gatling.io/>)

To understand which tool satisfies our needs, additional detailed information is required about each tool. To synthesize that information, a comparative study was carried out between the four tools. For the accomplishment of this study was necessary to evaluate the functionalities provided by the tools, its documentation and existing usage examples. It was also necessary to install each tool to ensure knowledge of its operation details. Throughout the next sections, four testing tools will be presented: Apache jMeter, Apache Flood, The Grinder and Gatling.

4.1 Apache jMeter

Apache jMeter [5] was designed to test functional load behavior, that is, to correctly simulate users and measure the applications performance. It can be used to test performance both in static resources as in dynamic resources (files, databases, server), to simulate a heavy server/group load, test the load limit supported by the server, or analyzing the overall performance under different kinds of load.

This tool simulates software usage by virtual users and simulates its web access simultaneously, generating data to delimit how many users manages to withstand before being cast away by its users. That is, Apache jMeter tests whether the expected number of users will be met within the timeframe described in the software requirements.

For the fulfillment of tests, Apache jMeter provides several kinds of requests and assertions, which authenticate the result of requests, besides that it also possesses logic controllers, such as cycles, and additional controllers to be used in the construction of test plans. Thread controls (virtual users) are also made available by this tool, designated as thread group, where it is possible to set the number of threads, the amount of times each thread is executed and the timeframe between each execution, all this assists in conducting stress tests. In the end, several listeners exist (tree, tables, graphs and log file), that based in the request results or in the assertions, can be used to generate graphs or tables results. One example of the results of a test in the form of a table are shown in Fig. 1.

View Results in Table

Name: View Results in Table

Comments:

Write results to file / Read from file

Filename Log/Display Only: Errors Successes

Sample #	Start Time	Thread Name	Label	Sample Time(...)	Status	Bytes	Latency	Connect Time(...)
1	12:47:40.356	FTP Users 1-1	File1	26	✔	24800	5	0
2	12:47:40.382	FTP Users 1-1	File2	84	✔	20131	4	0
3	12:47:40.466	FTP Users 1-1	File1	24	✔	24800	4	0
4	12:47:40.490	FTP Users 1-1	File2	23	✔	20131	4	0
5	12:47:40.607	FTP Users 1-2	File1	25	✔	24800	4	0
6	12:47:40.632	FTP Users 1-2	File2	25	✔	20131	5	0
7	12:47:40.657	FTP Users 1-2	File1	26	✔	24800	4	0
8	12:47:40.684	FTP Users 1-2	File2	23	✔	20131	3	0
9	12:47:40.857	FTP Users 1-3	File1	32	✔	24800	3	0

Fig. 1. Example of a test in Apache jMeter.

Apache jMeter initially works with the user sending a request to the destination server. Apache jMeter then receives the useful information from the destination server and provides the user with the results in different formats. The main features of Apache jMeter are:

- Can run on any operating system, since it is an application developed in JAVA;
- Supports HTTP, SMTP, POP3, LDAP, JDBC, FTP, JMS, SOAP and TCP protocols;
- Has multiple built-in and external listeners to view and analyze performance test results;
- Integration with major configurations and continuous integration systems is possible.

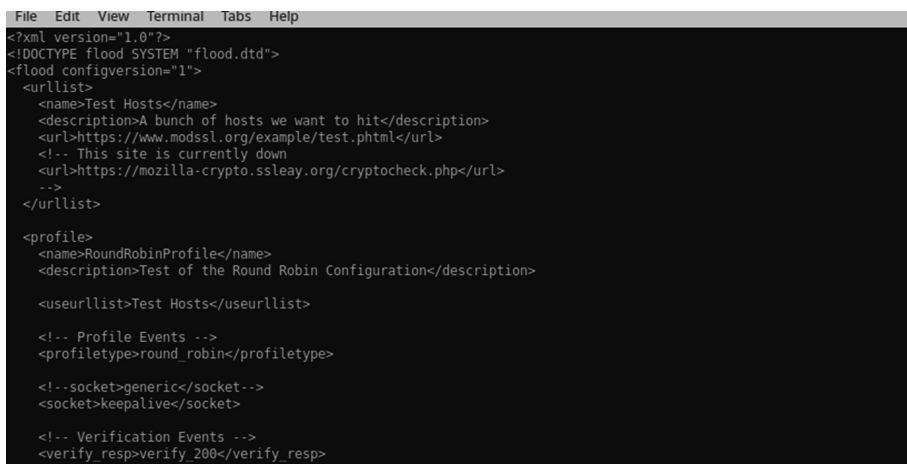
In summary, Apache jMeter is a tool that behaves completely in performing tests, since it supports static and dynamic resources, as well as several protocols from HTTP to TCP. It can even be used by any machine since it can run on any operating system. The fact of supporting distributed testing facilitates the simulation of a larger load, since each test runs on different machines.

4.2 Apache Flood

Apache Flood [4] can be used to collect performance metrics that are important for a given website, such as the time to submit an order or the time to receive a complete response.

It has a modular architecture where we can easily add new resources, designing frameworks through a support library and defining actions and behaviors for them. It is capable of generating large amounts of web traffic and of working well with dynamic content, being possible to simulate multiple, different and complex users, since a user can make a request to a more complete web page.

All tests are called with a standard interface and at the end transaction reports are collected with statistics of each HTTP transaction about the latency time, response time, idle time, and TCP Handshake time. A test example with Apache Flood is shown in Fig. 2.



```
File Edit View Terminal Tabs Help
<?xml version="1.0"?>
<!DOCTYPE flood SYSTEM "flood.dtd">
<flood configversion="1">
  <urllist>
    <name>Test Hosts</name>
    <description>A bunch of hosts we want to hit</description>
    <url>https://www.modssl.org/example/test.phtml</url>
    <!-- This site is currently down -->
    <url>https://mozilla-crypto.ssleay.org/cryptocheck.php</url>
    <!-- -->
  </urllist>

  <profile>
    <name>RoundRobinProfile</name>
    <description>Test of the Round Robin Configuration</description>

    <useurllist>Test Hosts</useurllist>

    <!-- Profile Events -->
    <profiletype>round_robin</profiletype>

    <!-- socket>generic</socket-->
    <socket>keepalive</socket>

    <!-- Verification Events -->
    <verify_resp>verify_200</verify_resp>
```

Fig. 2. Test Plan using Apache Flood [18].

With Apache Flood it is possible to run several tests in parallel and two options are provided for this execution: Threaded and Forked. These two methods can be used simultaneously, allowing to maximize the performance of each test. In threaded execution, the process is instructed to perform several user-space threads, each of which will execute a chain of complex events. When a Forked run is performed, the process is instructed to make multiple copies of itself using the fork() command.

Apache Flood makes it possible to perform distributed tests through access to several remote machines. It is possible to invoke a remote instance with both RSH and SSH.

The main features of Apache Flood are:

- Works with a global standard in security technology (SSL);
- Has an XML-based configuration;

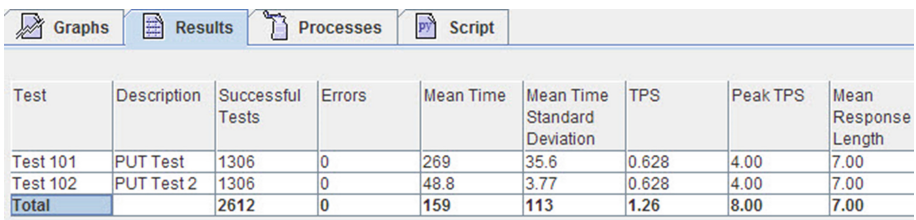
- Simulator of users and multiple users simultaneously;
- Simulator of several different users, that is, each with different arrival times.

Overall, Apache Flood is a little complex tool, since its configuration is only based on XML and works only with SSL. Besides that, it has little documentation, which makes it difficult to use.

4.3 The Grinder

The Grinder [23] is a load testing platform, developed in JAVA that makes it easy to run a distributed test. That is, it is possible to simultaneously use several machines to perform parallel tests, controlling all execution from a main machine, where we can generate various analyzes of the generated data, with tables and graphs. Each load to be monitored and controlled is visualized through a graphic console. It allows see how the application behaves under a heavy load, subsequently determining all weaknesses of the application in order to optimize them. The Grinder comes with a plugin for HTTP testing services as well as a tool that allows HTTP scripts to be automatically recorded.

The Grinder consists of agents, which initiate the number of load processes equal to the number configured by the user; by workers, who execute the load test scripts; by a console that is the graphical interface used to control the agents and to display the statistics collected by the workers and by TCP proxy that interposes between the browser and the destination server and can be used to create scripts by registering the activity of the browser, which can later be executed by work processes. Results of a test, in the form of a table, are presented in Fig. 3.



Test	Description	Successful Tests	Errors	Mean Time	Mean Time Standard Deviation	TPS	Peak TPS	Mean Response Length
Test 101	PUT Test	1306	0	269	35.6	0.628	4.00	7.00
Test 102	PUT Test 2	1306	0	48.8	3.77	0.628	4.00	7.00
Total		2612	0	159	113	1.26	8.00	7.00

Fig. 3. Result of a test using The Grinder.

The Grinder's main features are:

- Uses a TCP proxy to record network activity in the test script;
- It is possible to perform distributed tests that adapt with the increase of the number of users;
- Using Python or Closure with any Java API allows creation or modification of better test scripts;
- Post-processing with full access to the results of the correlation and content verification tests;
- Supports multiple protocols: SOAP, XML-RPC, IIOp, RMI/IIOp, RMI/JRMP, JMS, POP3, SMTP, FTP and LDAP.

Briefly, The Grinder is a tool that supports several protocols, from SOAP to LDAP. This allows the exchange of structured information on a decentralized and distributed platform to be carried out in several ways, some faster and more efficient than others. Besides, it is highlighted in its reports that allows an easy analysis of the test results, since they can be presented from tables to graphs.

4.4 Gatling

Gatling [8] was designed to be used with load testing, analyzing and measuring the performance of a variety of services, focusing on web applications, defending ease of use, maintenance and high performance. Gatling is written in Scala that comes with an interesting premise of always treating our performance tests as production code, meaning we can write code directly in the application. It is a very useful tool when we want to simulate a large number of users, since they do not all arrive at the same time and Gatling has an option (ramp) to implement this behavior, where the ramp value indicates the duration during which the users are started linearly, that is, they are always started for a fixed number of seconds. It also allows to simulate various types of users and even all these users using the application simultaneously.

Basically the Gatling structure can be defined in four different parts: (i) configuration of the HTTP protocol, where it is possible to define the base url to be tested; (ii) definition of headers, which makes it possible to add a bit of load through them to the server; (iii) definition of the scenario, which constitutes the core of the test, where a set of actions is performed to simulate a user interaction with the application and (iv) simulation definition, where is defined the load that will be executed over a period of time.

Gatling provides a diverse form of representation of results, as shown in Fig. 4.



Fig. 4. Result of test using Gatling.

The main characteristics of Gatling are:

- Easy integration with Jenkins through the jenkins-plugin and can also be integrated with other continuous integration tools. This allows constant feedback from performance tests;
- Allows to easily run the tests through Maven to Gradle with the help of maven-plugin and gradle-plugin;
- Full HTTP protocol support and can also be used for JDBC and JMS load tests;
- Has multiple input sources for data-driven testing.

In summary, Gatling stands out for the elegant reports it provides, as well as for the documentation it has about the operation of the application. It also has a very intuitive interface. The fact that its structure has the possibility of being defined in four different parts, allows the addition of charge even in headers.

5 Comparison of Web Platform Testing Tools

In this section we present a comparison of the four web platform test tools, and then a discussion of the results. This comparison is useful for users to choose the testing tool best suited to their needs. The comparison of the tools is divided into two analyses: qualitative and quantitative.

5.1 Qualitative Analysis

In order to perform the qualitative analysis, only the most relevant characteristics were considered and, after the use of each of the tools, each one of its characteristics was described. The characteristics analysed are: Open Source, Report view mode, Test language, Test recorder, and Distributed testing. Table 2 shows the qualitative analysis of the four tools.

Table 2. Qualitative analysis.

Characteristics	Tools			
	Apache jMeter	Apache Flood	The Grinder	Gatling
Open source	Yes	Yes	Yes	Yes
Report view mode	CSV, XML, Embed. Tables, Graphic, Plugins	CSV, XML, Embed. Tables, Graphic, Plugins	CSV, XML, Embed. Tables, Graphic, Plugins	CSV, XML, Embed. Tables, Graphic, Plugins
Test language	XML	XML	XML	XML
Test Recorder	HTTP	HTTP	HTTP	HTTP
Distributed Testing	Supports multiple machines to be controlled by a single instance to run	Supports multiple machines to be controlled by a single instance to run	Supports multiple machines to be controlled by a single instance to run	Supports multiple machines to be controlled by a single instance to run

In this analysis the Apache jMeter tool stands out as being the best one, since it has more visualization modes than the other tools. Regarding distributed testing this tool is also better, since it allows the use of multiple machines to be controlled by a single instance for execution.

5.2 Quantitative Analysis

In order to perform the quantitative analysis, only the most relevant characteristics were considered: ease of use, graphics complexity, interface, quality of documentation presented, easily editing scripts, and ease of interpretation of reports. After each tool use, each one of its characteristics was evaluated on a scale from zero (very bad) to ten (excellent). Then the individual scores of each feature of the respective tools were added, resulting in their final score.

In Table 3 the quantitative analysis of the four tools is performed.

Table 3. Quantitative analysis [18].

Characteristics	Tools			
	Apache jMeter	Apache Flood	The Grinder	Gatling
Ease of use	10	4	6	5
Graphics complexity	8	0	7	6
Interface	9	2	8	7
Quality of documentation presented	10	2	8	6
Easily editing scripts	10	3	7	6
Ease of interpretation of reports	8	5	4	3
Final score	55	16	40	35

The final scores obtained by the Apache jMeter, Apache Flood, The Grinder and Gatling were respectively: 55, 16, 40, and 35 points.

The jMeter tool was considered to have a better ease of use, since it has an intuitive interface and a lot of documentation about its use. In turn, Apache Flood was the tool with a lower ease of use, since it does not have an interface, nor does it have documentation about its operation. Regarding the ease of editing scripts, the jMeter tool is the “winner”, since it is not necessary to write direct code to perform editing. The more complex graphs of jMeter lead to the production of more detailed reports with a high ease of interpretation.

The tool with the lowest score was Apache Flood, since it had a negative score in practically all analyzed characteristics. With regard to the best score, this was attributed to the jMeter tool, since it stood out positively in all the characteristics, never obtaining a score inferior to 8.

5.3 Discussion of the Results

After enumeration and analysis of the four test platforms, it was necessary to choose the ones that would allow easy learning, be simple to handle, be able to simulate several users on the website, send requests to the server, support distributed testing and generate reports in CSV. That is, it was necessary to select a tool that pleases the user in all its aspects, from its use, to the results visualization. So, the selected tools were Apache jMeter and The Grinder. Apache jMeter stands out positively in relation to the mentioned characteristics and have a good documentation in its own website that originates a good learning of use is evidenced by the examples that it has in the most diverse tasks that allow to create test scripts in a simplified way.

The fact that jMeter supports distributed testing, with multiple machines simultaneously being controlled by a single running instance allows test scripts with a small number of users to run on multiple machines. Thus it is possible to simulate a greater number of users in the same time interval, thus avoiding long execution times for the various tests performed.

The second best tool selected was The Grinder, as it stands out in its user-friendly interface and documentation. It is also possible to construct more complex test plans, since they are written with code and not created through the tool console.

6 Performance Testing with Web Search Engines

Performance testing is performed to evaluate the performance of components of a particular system under a particular workload. During this testing, system components are monitored to verify the stability of the system under test. Performance Testing is the type of Non-Functional Testing. It strives to build performance standards into the implementation, design and architecture of a system.

The speed of a website is as important today as its content, because unconsciously no one likes to wait, so every millisecond matters a lot in response time. Smartphones, tablets and other portable devices are creating more and more web traffic simultaneously, since countless people download videos, news or use social networks. Therefore, there is a huge competition not through the content that each website has, but because of its responsiveness. Certainly the most visited is the one that will respond faster. Google engineers have revealed that users are starting to get frustrated with a website after waiting only 400 ms [13], this almost imperceptible delay causes users to look for other websites.

According to [20] we select the top five web search engines: Google, Bing, Yahoo, Ask and AOL Search. Of these five, Yahoo was excluded, and the others were compared in terms of performance to the remaining search engines by using the jMeter tool. Yahoo was removed from the comparison, since whenever users were simulated to access it, an error was returned with code 999 (unable to process request at this time). This error occurs when there is a large number of requests originating from the computer that is being used to perform the tests. Yahoo generates error code 999, to protect its servers, denying access to its webpage [10].

In order to perform the tests, we used the two best tools, Apache jMeter and The Grinder, respectively, so we could check if there were differences between the results returned by each of the tools. The test case consisted of adding more words in the search as well as gradually increase the number of users in each web search engine. Thus, searches were done with the words “Apache jMeter”, “Apache jMeter Gatling” and “Apache jMeter Gatling Grinder”.

In the experiments we simulate 10, 100, and 1000 users accessing the web page. All users were simulated coming to the search engines every 5 s, since they always have countless people accessing their web pages. Only 10 users were initially simulated to understand how each search engine behaved with a minimum load. This number has been increasing to an average load with 100 users and to a high load with 1000 users. Since 1000 users already experienced errors in simulating users, this number was no longer increased.

For testing the web search engines, we evaluate two metrics:

- Elapsed time: it encompasses the time from the time the request was formulated, until its response was obtained.
- Message size in bytes: it incorporates the size of customer requests for server and server responses to the client.

Each test case was repeated ten times and only the arithmetic mean of the results obtained are presented in Tables 4, 5 and 6. We also computed the latency time (it only includes the time from when the request was formulated until the first part of the response was obtained) but the results are similar to elapsed time and due to space limits they are not shown.

Table 4. Elapsed time tests performed with Apache jMeter.

	Search Engine	Elapsed Time (ms)		
		2 words	3 words	4 words
10 users	Google	396	378	2,080
	Bing	575	1,569	4,067
	Ask	802	3,009	3,109
	Aol Search	441	1,060	1,060
100 users	Google	443	350	1,125
	Bing	2,201	2,853	1,167
	Ask	1,171	24,318	24,318
	Aol Search	731	13,108	13,108
1000 users	Google	11,525	12,835	16,166
	Bing	43,354	60,242	81,904
	Ask	20,225	28,951	30,327
	Aol Search	19,776	12,510	12,907

Table 5. Message size with Apache jMeter.

	Search Engine	Message size (bytes)		
		2 words	3 words	4 words
10 users	Google	40,686	40,817	41,982
	Bing	91,820	93,013	93,626
	Ask	71,390	134,936	104,704
	Aol Search	48,417	40,407	40,212
100 users	Google	40,656	40,810	41,986
	Bing	91,932	93,156	93,774
	Ask	71,411	134,885	101,708
	Aol Search	48,417	40,405	40,212
1000 users	Google	40,623	40,869	41,801
	Bing	91,495	92,853	102,12
	Ask	35,804	87,902	85,926
	Aol Search	48,417	40,404	40,546

Table 6. Elapsed time tests performed with The Grinder.

	Search engine	Elapsed Time (ms)		
		2 words	3 words	4 words
10 users	Google	420	505	653
	Bing	2,000	2,100	3,450
	Ask	5,400	6,100	7,880
	Aol Search	245	328	2,210
100 users	Google	560	682	1,980
	Bing	2,300	3,400	4,600
	Ask	1,400	2,180	1,870
	Aol Search	250	482	1,200
1000 users	Google	580	23,300	2,940
	Bing	2,700	26,500	34,000
	Ask	1,800	46,300	54,400
	Aol Search	550	27,300	29,900

6.1 Results with the Apache jMeter

In this section we present and analyze the results of the tests performed with the best tool, Apache jMeter. The results refer to tests with two, three and four search words.

According to the results presented in Table 4, the search engine that takes less time to obtain the response since the request was made is Google, searching 2 words. So it responds quickly to its users, avoiding waiting queues, because with 10 and 100 users the elapsed values are less than 450 s. When the load is increased to 1000 users, it also increases the elapsed time. In this case it is possible that some users wait longer than others to obtain the response to their request.

With the increase in the number of words to search, also increases the time to get the answer since the request was made. Overall, the best search engine when more than two words are used is Google, since it has lower elapsed time values. A good alternative to Google in terms of performance will be Aol Search, which also has reduced elapsed values.

In contrast, the search engine that responds more slowly to its users by providing higher elapsed values is Bing. This one with 100 and 1000 users can have elapsed values that represent twice the value obtained in the other search engines.

According to the results presented in Table 5, the amount of data returned in the server response (message size) is practically constant with increasing load, since the search results are the same. In turn, when we increase the number of words to search, the size of the message increases accordingly, since the returned links will be different when searching for different words. Google is the search engine that returns fewer bytes in its response, that is, it responds faster to the user. A good alternative to Google is Aol Search, since it also returns minimum message sizes. In contrast, the search engine that responds more slowly is Bing.

6.2 Results with the Grinder

In this section are presented and analyzed the results of the tests performed with the tool The Grinder. The results refer to tests with two, three and four search words.

According to the results Table 6, the search engine that responds more quickly when a search is performed on it, is Google. Despite the increase in the number of words the search also increases the response size, it can respond quickly when it has a low charge. When the load and number of words are increased simultaneously, Google responds to respond more slowly, however it continues to stand out positively in response times.

An alternative to Google is Aol Search, which behaves very similarly to its elapsed time values.

In contrast, search engines that have returned the worst results, resulting in multiple queues when they have a high load on the system, are Bing and Ask.

The results presented in Table 7 behave similarly to those in Table 5, that is, the number of bytes returned in the response remain practically constant with the increase in load. However, when the number of words being searched is increased, so does the size of the message.

Despite similar behaviors, the values in Table 7 differ greatly from those in Table 5. Apache jMeter does not perform all actions supported by browsers, in particular it does not perform the javascript found in HTML pages. Thus, Table 5 presents lower values for the size of the message.

However, despite different values, the conclusions drawn are the same: Google is the fastest web search engine, followed by Aol Search and, in turn, Bing responds more slowly, since it returns more Bytes in the response.

Table 7. Message size with The Grinder.

	Search engine	Message size (bytes)		
		2 words	3 words	4 words
10 users	Google	43,200	45,000	45,600
	Bing	97,300	97,700	97,900
	Ask	82,980	86,400	86,800
	Aol Search	53,200	54,000	57,500
100 users	Google	43,400	45,800	45,600
	Bing	97,100	97,000	99,800
	Ask	88,500	80,600	86,800
	Aol Search	56,700	57,900	54,000
1000 users	Google	43,500	44,800	43,800
	Bing	96,000	98,500	91,000
	Ask	85,800	87,000	89,200
	Aol Search	52,300	52,000	53,400

7 Accuracy of Results Using Web Search Engine

As important as the speed with which a particular search engine responds is the accuracy of the results returned by it. Thus, five questions were conducted where the response was studied according to the number of correct results (accuracy). We analyzed the results returned on the first five pages returned in the search (#1, #2, #3, #4, #5) and on the fifteenth page (#15) to see if the search engine continued to show results identical to those on the original pages. The number of correct results returned in each search engine was recorded, thus concluding which one had better accuracy in the search results.

We choose the following four questions to test the web search engines:

- Where is ISEC?
- Where is Rua Pedro Nunes?
- Who is the Portuguese Prime Minister?
- Who was the last King of Portugal?

According to the results presented in Table 8, the search engine that returns a larger number of links related to a specific search is Google, leading in the polls “Who is the Portuguese Prime Minister?” and “Who was the last King of Portugal?”. However, considering only the number of correct results in the first web page (#1), the best web search engine is Bing, since it returns a larger number of results on the first page. In contrast, Aol Search is the one that returns the least correct links.

Google, Bing, and Ask are the web search engines that return a significant amount of correct links on the first four pages. In turn, Aol Search distributes the total number of correct links throughout all pages, since it returns the same amount on almost every page.

So the best search engine in terms of correct results returned related to the search questions is Google.

Table 8. Number of correct links returned in each web search.

Questions researched	Search engines	Page number						Total
		#1	#2	#3	#4	#5	#15	
“Where is ISEC?”	Google	1	0	0	0	0	0	1
	Bing	8	9	8	2	2	0	29
	Ask	1	0	0	0	0	0	1
	Aol Search	0	0	0	0	0	0	0
“Where is Rua Pedro Nunes?”	Google	1	2	2	2	0	0	7
	Bing	2	1	3	2	0	0	8
	Ask	0	5	0	7	1	0	13
	Aol Search	1	0	1	3	3	3	11
“Who is the Portuguese Prime Minister?”	Google	7	9	6	8	2	3	35
	Bing	3	1	2	3	2	5	16
	Ask	6	5	4	4	5	3	27
	Aol Search	3	5	1	4	5	3	21
“Who was the last King of Portugal?”	Google	6	5	8	4	3	0	26
	Bing	3	0	2	0	2	2	9
	Ask	6	4	2	2	3	0	17
	Aol Search	5	4	4	2	1	0	16
Total per page	Google	<i>15</i>	<i>16</i>	<i>16</i>	<i>14</i>	<i>5</i>	<i>3</i>	69
	Bing	<i>16</i>	<i>11</i>	<i>15</i>	<i>7</i>	<i>6</i>	<i>7</i>	62
	Ask	<i>13</i>	<i>14</i>	<i>6</i>	<i>13</i>	<i>9</i>	<i>3</i>	58
	Aol Search	<i>9</i>	<i>9</i>	<i>6</i>	<i>9</i>	<i>9</i>	<i>6</i>	48

8 Conclusions and Future Work

Performance testing strives to build performance standards into the implementation, design and architecture of a system. The activity test is fundamental to guarantee the quality of the products developed. Among the various types of test that exists, load testing stands out, since they are reaching more and more importance, since the web systems are increasingly used. These tests are still little used and, since the associated costs for their execution are high, the use of tools that automate the creation and execution of tests is essential. In addition, measuring response time and simulating multiple users accessing an application simultaneously is unfeasible and often impossible to perform without a tool that automates the testing process.

In this paper, four test tools were presented: Apache jMeter, Apache Flood, The Grinder and Gatling. For this, a quantitative analysis and a qualitative analysis were carried out, comparing the main characteristics common to all the tools. This comparison concludes that the best tool for evaluation of web platforms is Apache jMeter, since it stands out positively in most of its characteristics, in addition to allowing the use of distributed tests, with multiple machines simultaneously to be executed by a single instance running. The fact that Apache jMeter supports distributed testing, provides the simulation of more users in the same time interval.

As search engines are accessed countless times during the day by various users around the world, it is important that they can respond quickly to all users. Thus, it is important to perform performance tests on the search engines, and then act on their optimization. In the experimental evaluation load tests were performed on four search engines with the Apache jMeter. To be able to evaluate different results, was used the second best tool for evaluation of web platforms, The Grinder. The number of search words was further increased to see if this number interfered directly with response times.

So the search engine that stood out positively both in terms of response times and in terms of accuracy was Google. This low load (10 users), average (100 users) or high (1000 users) responds quickly to users. In contrast, Bing was the search engine that stood out negatively, since it is the one that takes more time to present the answers to the users. With high response times users tend to abandon their page.

In terms of accuracy, the most accurate search engine in searches performed is Google, since it returns many links related to the searched words. This is more efficient in your results on the first four pages. An alternative to Google in terms of accuracy is Bing, which also returns a significant amount of search-related links. In contrast, the search engine that returns the least results related to the search is Aol Search, however it distributes the correct links evenly through the results pages. So the best search engine in terms of accuracy and performance is Google. A good alternative to this in terms of performance is Aol Search, however in terms of accuracy is Bing.

As future work we propose the creation of distributed tests, that is, in several machines. This way, we can simulate more users accessing the same search engines already tested and get more real results.

References

1. Avritzer, A., Weyuker, E.R.: The automatic generation of load test suites and the assessment of the resulting software. *IEEE Trans. Softw. Eng.* **21**(9), 705–716 (1995). <https://doi.org/10.1109/32.464549>
2. Avritzer, A., Larson, B.: Load testing software using deterministic state testing. In: Ostrand, T., Weyuker, E. (eds.) *Proceedings of the 1993 ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 1993*, pp. 82–88. ACM, New York (1993). <https://doi.org/10.1145/154183.154244>
3. Avritzer, A., Weyuker, E.J.: Generating test suites for software load testing. In: Ostrand, T. (ed.) *Proceedings of the 1994 ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 1994*, pp. 44–57. ACM, New York (1994). <https://doi.org/10.1145/186258.186507>
4. Apache Flood. <https://httpd.apache.org/test/flood/>. Accessed 11 Nov 2016
5. Apache JMeter. <http://jmeter.apache.org>. Accessed 11 Nov 2016
6. Difference Between Performance Testing, Load Testing and Stress Testing – With Examples. <http://www.softwaretestinghelp.com/what-is-performance-testing-load-testing-stress-testing/>. Accessed 15 July 2017
7. Wang, F., Du, W.: A test automaton framework based on WEB. In: *Proceedings of the IEEE 11th International Conference on Computer and Information, ACIS 2012*. IEEE Press (2012)
8. Gatling Project, Stress Tool. <http://gatling.io>. Accessed 11 Nov 2016

9. Banga, G., Druschel, P.: Measuring the capacity of a web server under realistic loads. *World Wide Web* **2**(1–2), 69–83 (1999). <https://doi.org/10.1023/A:1019292504731>
10. Information about Yahoo Error 999. <http://www.scrapebox.com/yahoo-999-error>. Accessed 11 Nov 2016
11. Zhang, J., Cheung, S.C.: Automated test case generation for the stress testing of multimedia systems. *Softw. Pract. Experience J.* **32**(15), 1411–1435 (2002). <https://doi.org/10.1002/spe.487>
12. Curran, K., Duffy, C.: Understanding and reducing web delays. *Int. J. Netw. Manag.* **15**(2), 89–102 (2005)
13. Lohr: For Impatient Web Users, an Eye Blink Is Just Too Long to Wait (2012). http://www.nytimes.com/2012/03/01/technology/impatient-web-users-flee-slow-loading-sites.html?_r=2
14. Bayan, M.S., Cangussu, J.W.: Automatic stress and load testing for embedded systems. In: 30th Annual International Computer Software and Applications Conference (COMPSAC 2006), Chicago, IL, pp. 229–233 (2006). <https://doi.org/10.1109/COMPSAC.2006.119>
15. Sharma, M., Angmo, R.: Web based automation testing and tools. *Int. J. Comput. Sci. Inf. Technol.* (2014)
16. Sharma, M., Iyer, V.S., Subramanian, S., Shetty, A.: A comparative study on load testing tools. *Int. J. Innovative Res. Comput. Commun. Eng.* (2007)
17. Barford, P., Crovella, M.: Measuring web performance in the wide area. *SIGMETRICS Perform. Eval. Rev.* **27**(2), 37–48 (1999). <https://doi.org/10.1145/332944.332953>
18. Paz, S., Bernardino, J.: Comparative analysis of web platform assessment tools. In: Proceedings of the 13th International Conference on Web Information Systems and Technologies - Volume 1: WEBIST, pp. 116–125 (2017). ISBN 978-989-758-246-2. <https://doi.org/10.5220/0006308101160125>
19. Pressman, R.: *Engenharia de Software*, 6th edn. McGraw-Hill, New York (2006)
20. Ratcliff (2016). <https://searchenginewatch.com/2016/08/08/what-are-the-top-10-most-popular-search-engines/>. Accessed 11 Nov 2016
21. Khan, R.: Comparative Study of Performance Testing Tools: Apache JMeter and HP LoadRunner (2013)
22. Hussain, S., Wang, Z., Toure, I.K., Diop, A.: Web Service Testing Tools: A Comparative Study (2013)
23. The Grinder, a Java Load Testing Framework. <http://grinder.sourceforge.net/>. Accessed 11 Nov 2016
24. Tikhanski: Open Source Load Testing Tools: Which One Should You Use? (2015). <https://www.blazemeter.com/blog/open-source-load-testing-tools-which-one-should-you-use>
25. Garousi, V., Briand, L.C., Labiche, Y.: Traffic-aware stress testing of distributed systems based on UML models. In: Proceedings of the 28th International Conference on Software Engineering (ICSE 2006), 391–400. ACM, New York (2006). <https://doi.org/10.1145/1134285.1134340>
26. Wang, X., Zhou, B., Li, W.: Model based load testing of web applications. In: Proceedings of IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA 2010). IEEE Press (2010)
27. Jiang, Z.M., Hassan, A.E., Hamann, G., Flora, P.: Automatic identification of load testing problems. In: Proceedings of the 24th IEEE International Conference on Software Maintenance (ICSM), Beijing, pp. 307–316 (2008). <https://doi.org/10.1109/ICSM.2008.4658079>
28. Jiang, Z.M.: Automated analysis of load testing results. In: Proceedings of the 19th International Symposium on Software Testing and Analysis (ISSTA 2010), pp. 143–146. ACM, New York (2010). <https://doi.org/10.1145/1831708.1831726>



Progressive Web Apps for the Unified Development of Mobile Applications

Andreas Bjørn-Hansen^{1(✉)}, Tim A. Majchrzak², and Tor-Morten Grønli¹

¹ Faculty of Technology, Westerdals Oslo ACT, Oslo, Norway
`{bioand,tmg}@westerdals.no`

² ERCIS, University of Agder, Kristiansand, Norway
`timam@uia.no`

Abstract. Progressive Web Apps (PWAs) allow for web applications to be developed in an offline-first approach. While Web apps traditionally did not properly function without an Internet connection, PWAs enable them to be downloaded, installed and used offline on various systems, including mobile devices and personal computers. We present an introduction to the state-of-art in research and practice. Based on this, we discuss various underlying concepts and technologies. Then, we scrutinize and compare PWAs against cross-platform app development approaches on both technical and overarching aspects. A list of suggestions for future research is also presented. We urge academia to keep up with the latest trends within the field of mobile and web development, as new concepts and approaches such as Progressive Web Apps now make for interesting research topics with tangible real-world effects.

Keywords: Progressive Web Apps · Service workers · Cross-platform Cross-platform development · Mobile web

1 Introduction

The proliferation of app- and web-enabled devices including mobile, watches, TVs, desktop and laptop computers demands developers to obtain considerable multi-platform technical knowledge due to extensive fragmentation. In traditional application development, codebases have been constrained to build and execute on only one or few target platforms [2–4]. In mobile development, we refer to this as the *Native* development approach – apps are developed in a platform-specific language such as Java or Kotlin for Android and Swift or Objective-C for iOS. The clash of business interests, where an increasing pool of devices and

This paper greatly extends the short paper [1] presented at WEBIST 2017. It has been updated to reflect the latest developments, includes new content based on additional work as well as on the discussions at the conference, and has been amended with a comprehensive discussion. Please note that verbatim content from the short paper is not explicitly highlighted but for figures and tables already included there.

platforms can run apps and display web content, yet pose development constraints to the developers' platform knowledge, has led to the increased popularity of tools and frameworks for cross-platform development [5].

The purpose of such tools and frameworks is to maximize code reuse across platforms and to minimize the product's time-to-market. Moreover, they allow efficient use of existing (or more common) knowledge [6,7] such as web development languages versus platform-specific mobile programming languages. Consequently, the vast amount of different cross-platform development frameworks has also been subject to academic research in recent years [6,8,9]. Different frameworks pose different possibilities, challenges and purposes. Research and practice tend to categorize such frameworks into *approaches* [10,11]. *Hybrid, Interpreted, Cross-compiled, Model-Driven (MDD)* and *Component-Driven (CDD)* development are all examples of cross-platform approaches [1,9,10]. These approaches differ in how they facilitate the development of cross-platform applications.

The Hybrid approach is well-known due to the popularity of frameworks such as Cordova, PhoneGap and Ionic, but has been criticised [9,12,13] for delivering sub-par performance and user experience when compared to native development due to its dependency on a WebView (an in-app integrated browser component) [9] and interfaces built in HTML and CSS. The Interpreted approach has seen a rise in popularity after the release of frameworks such as Facebook's React Native, Telerik's NativeScript and Fusetools' Fuse, none of which are dependent on a web browser, but rather leverage an on-device JavaScript interpreter to generate *native* user interfaces for an optimal user experience [14]. The Cross-compiled approach depends neither on web browsers nor interpreters, as it compiles a common language into native binaries for each supported platform, thus producing actual native apps [15]. Xamarin is an example of a tool belonging to the Cross-compiled approach [14]. Both the Model-Driven and Component-Driven approaches have previously been identified to serve mostly academic purposes, with less foothold in the industry than the aforementioned approaches [8,14].

The main differentiators between the Web platform and the aforementioned approaches and frameworks is that any cross-platform approach will produce apps that (a) Can access most or all open device and platform feature APIs, (b) Can be installed on users' phones, and (c) Do not require an active Internet connection to be used. These differences are some of several motivations behind a new Web-based development approach, coined **Progressive Web Apps (PWA)** [16]. Any website can become a Progressive Web App by adhering to and adopting certain concepts and technologies – as further discussed in detail in Sect. 4.

At its core, PWA allows websites (web apps) to become less *web-like* and more *app-like*. This is done by e.g. allowing websites to be downloaded and installed onto users' phones, similar to how regular apps are installed via app stores (example given in Fig. 7). Due to being installable, a PWA can – and should – also work well in offline contexts by leveraging browser-based storage engines such as IndexedDB [17]. This can be immensely important for emerging markets where Internet connectivity is sparse and expensive [18]. The mobile-web unifying part of PWAs also lies in how they *look* like regular apps.

We have previously reported about a lack of research on Progressive Web Apps [1]. This was true in January 2017 when researching towards our WEBIST short paper [1], and it still holds true as of July 2017. Little additional research has been added to the body of knowledge since our last literature review. We hope that by providing an array of suggestions for future work (see p. 18), researchers from different fields may take interest in further widening the area of research within Progressive Web Apps.

The rest of this paper is structured as follows. Section 2 explains our research methods, being literature search, technical implementation and performance testing. In Sect. 3 we discuss related work on cross-platform development and Progressive Web Apps. Underlying technologies and concepts are introduced in Sect. 4. Then, in Sect. 5 we present our findings, both a feature comparison of approaches, as well as a performance measurement comparison of the implemented technical artifacts. We then discuss our findings in Sect. 6, before we conclude our paper and suggest topics for further research in Sect. 7.

2 Research Methods

Our study implements three research methods to gain broader understanding of the possibilities and state of research and practice regarding Progressive Web Apps. We conduct a literature search to understand the state of the knowledge body, develop five technical implementations in the form of mobile apps, then scrutinize and compare the performance of those apps. This section discusses the methods in greater depth.

2.1 Literature Search

Our searches for academic involvement in Progressive Web Apps returned a very limited amount of results per July 2017, as it did during our initial search in January 2017 [1]. A search for “Progressive Web Apps” on Google Scholar returned 14 results. Of those, relevant academic results accounted for only three; a study from Malavolta et al. on the energy impact of service workers [19] (*ACM*), a paper regarding web application launch time [20] (*IEEE*), and one Bachelor’s thesis [21] investigating differences between PWAs and native Android apps. Also, a ResearchGate search was conducted, covering both publications and questions on their website, but it rendered no published results.

Academic contributions are, per July 2017, still close to non-existing. The lack of involvement is understandable as the subject is rather new and unexplored by the majority of research and practice. However, this results in a massive research potential within a variety of different fields, particularly technical, economical and sociological. Our suggestions for further research will hopefully spark interest in relevant research communities.

2.2 Technical Implementation

To gain better understanding of the possibilities of Progressive Web Apps, four apps and a PWA were developed for comparison. A *native* Android app was developed using the newly-supported Java alternative Kotlin [22]. An *hybrid* app was developed using the Ionic Framework. An *interpreted* app was developed using React Native. A *cross-compiled* app was developed using Xamarin. Lastly, the *Progressive Web App* was developed using React.js. As our main aim is to introduce PWA as a concept along with technical and higher-level comparisons, we excluded the aforementioned approaches *model-driven* and *component-driven* from our study, also due to the reported academic nature of said approaches with possibly limited real-world impact so far (cf. e.g. [23]).

The applications employ a master-detail navigation pattern, where the master view presents a list of clickable items fetched from an API [24]. Upon list item click, the app navigates to the detail view, displaying the item image together with its author and title. The data is fetched from www.reddit.com/r/Art/.json.

Note that no code- or otherwise technical *optimizations* have been implemented in either apps. All apps do however implement asynchronous data fetching to avoid blocking the user interface rendering. The apps are depicted in Figs. 1, 2, 3, 4 and 5. All five technical artefacts have been open-sourced¹ to allow verification of the results.

2.3 Performance Testing

Three variables were included as part of the performance test. The tests were conducted on a Google Nexus 5X device running Android version 7.0. The aim was to gather initial data on differences in performance between the implementations discussed in Subsect. 2.2.

Size of Installation: Information regarding app size was found at the following path on the Android device: **Settings** – **Apps** – [*app name*].

Activity Launch Time: This was measured using the *Android Debug Bridge* and its `am_activity_launch_time` command. This measures the time it takes to launch the first activity of an application. The test was conducted ten times for each app as the results rendered different at each run. The average time is presented as result.

Time from App-icon Tap to Toolbar Render: As no suitable method for measuring elapsed time from *app-icon tap* to user interface *toolbar render* was identified, an online stop watch was used. The toolbar, titled “Art by Reddit”, is displayed in Figs. 1, 2, 3, 4 and 5. In our applications, this element is the first to be rendered to the screen as it does not require any dynamic data to load (unlike the Reddit posts requiring an asynchronous API call), thus we used it as the basis for our render-time measurement. As the measurement relied on human reactions, and is thus inherently error-prone, the test was conducted ten times for each app, and the average time is presented as result.

¹ <https://github.com/andreasbhansen/pwa-paper>.

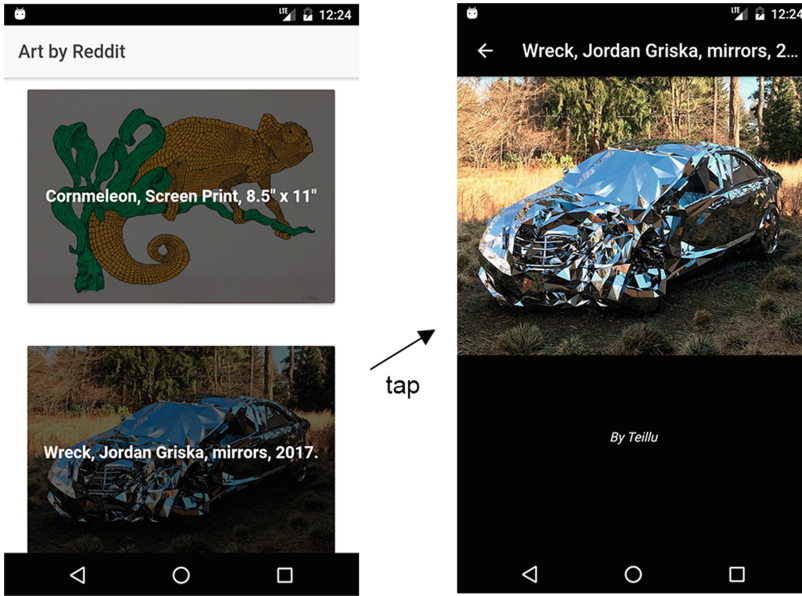


Fig. 1. The Ionic Framework Hybrid app.

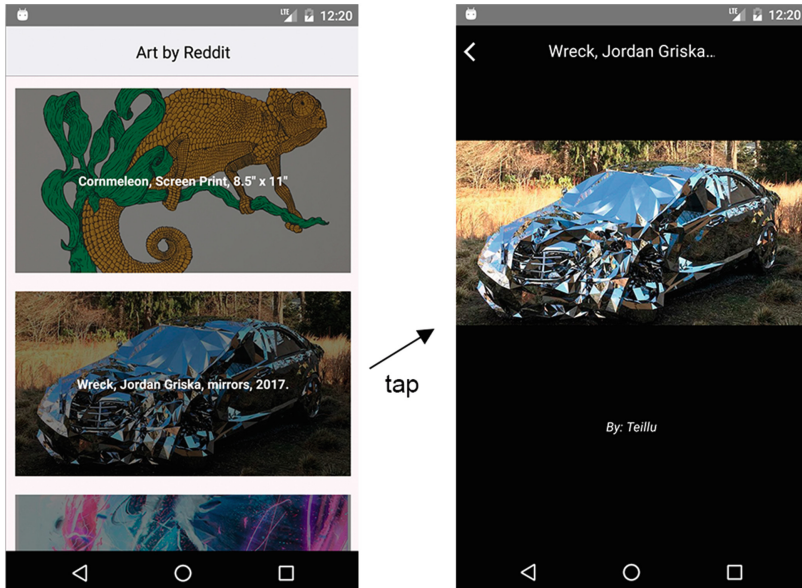


Fig. 2. The React Native Interpreted app.

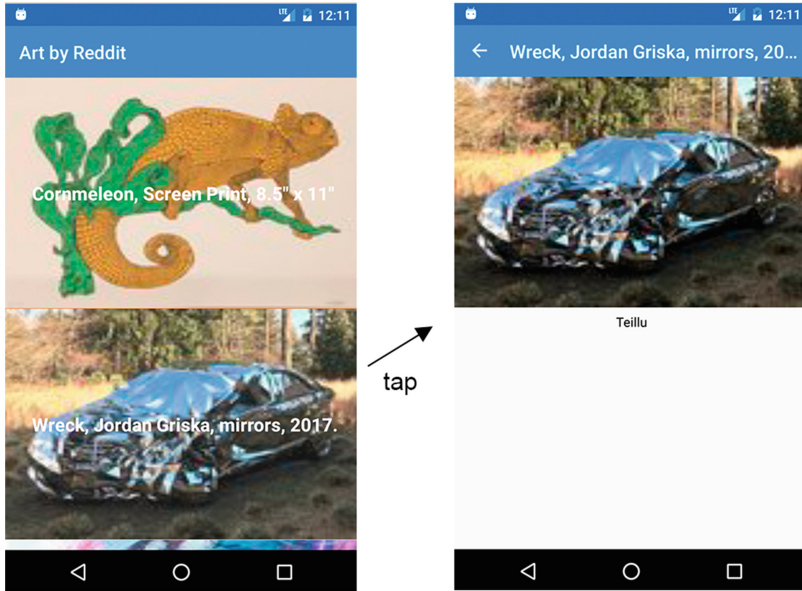


Fig. 3. The Xamarin Cross-Compiled App.

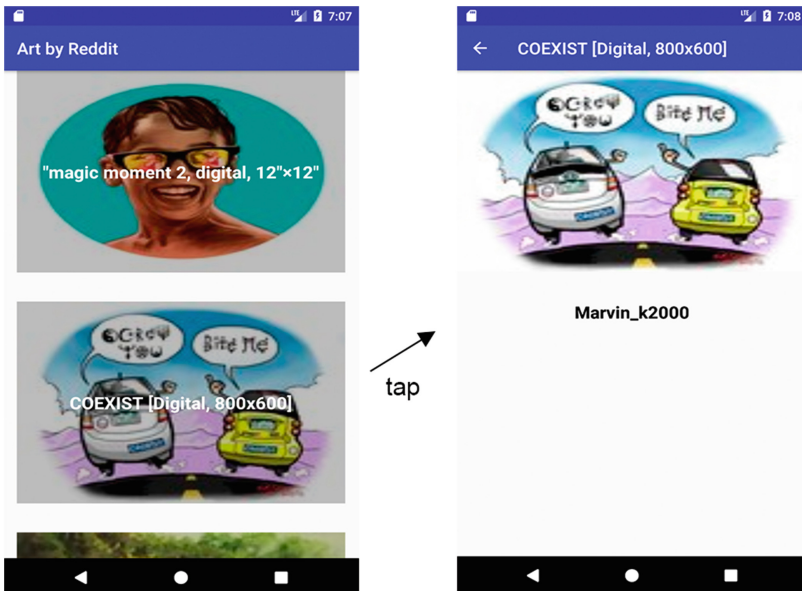


Fig. 4. The Native Android App.

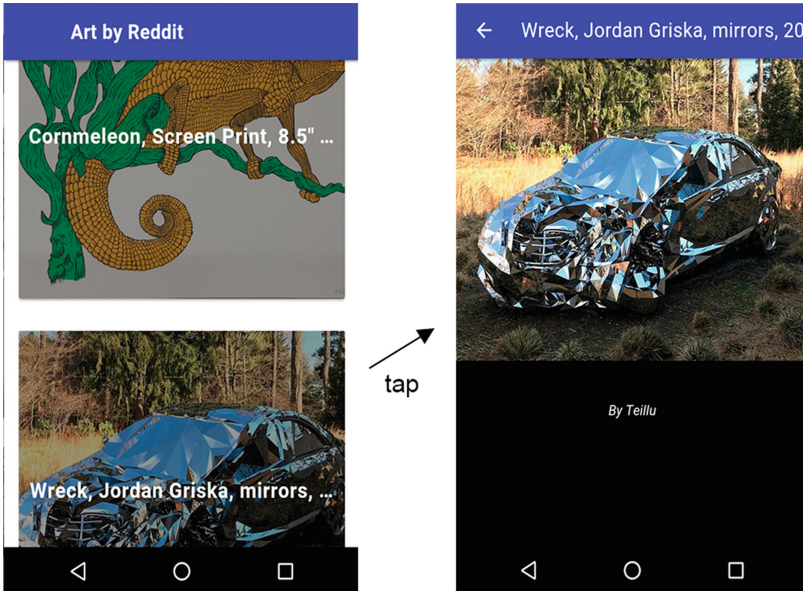


Fig. 5. The Progressive Web App.

3 Related Work

While the lack of academic involvement and research on Progressive Web Apps is evident from the literature search, an established industry-oriented body of knowledge was identified. The *Google Web Fundamentals* group acts as the driving force for the creation and publishing of tutorials and blog posts. Until a solid academic knowledge-base is developed, the Web Fundamentals website should, thus, be the foundation for upcoming research.

Other than Google-created content, two early-access books were identified [25,26] together with an extensive pool of industry articles. Examples of topics covered are fundamental concepts [27], challenges and concerns [28], and thoughts on the impact of web innovation [29]. A notably larger academic foundation was found for the other app development approaches discussed. Numerous papers on cross-platform app development and underlying approaches have previously been identified. Papers found to often recur in related research includes [7,30–32]. They make up the theoretical foundation and early research, which newer papers draw from. Their topics range from comparisons of technical frameworks and approaches, to classifications and performance measurements.

Recent research identified includes, but is not limited to, topics such as requirements for cross-platform tooling [33], development approach evaluation frameworks [11], energy consumption comparisons [15], and end-user perception of cross-platform apps [34,35]. Their contributions should be acknowledged as important foundation for future research on Progressive Web Apps.

Due to the lack of academic contributions regarding PWA, the keen industry interest should act as a catalyst for further research and more academic involvement. Research suggestions are presented in Sect. 7 in an attempt to spark interest in relevant research communities.

4 Underlying Technologies and Concepts

Progressive Web Apps are not a single technology or concept, but rather a variety of these used together to provide an improved user experience. In this section, we introduce different aspects of PWAs.

4.1 Regarding the Term

A *Progressive Web App* still is *web app*, although it uses techniques to provide a *progressive* user experience. It, therefore, is no inconsistency to speak of both *Progressive Web Apps* and *progressive web app* without addressing the same concepts. Arguably, an PWA is always a *progressive web app*, but not all *web apps* that are *progressive* are PWAs in a narrow sense. This confusion will be resolved by explaining the technologies and concepts in the following subsections.

Throughout this paper, we use the upper-case variant and respectively the acronym *PWA* when we refer to the agglomeration of technologies and concepts as a whole, i.e. when an app qualifies to be a PWA. Lower case variants are used when concepts are addressed more loosely, e.g. when an app embeds some concepts of progression.

We deem this distinction to be important; for the time being, it should be sufficient as proposed above. However, with a possible proliferation of PWAs a clear denomination will be required (cf. also with the call for a taxonomy of app-enabled devices in [36]).

4.2 Technologies

Service Workers. Most of the features and concepts (see Sect. 4.3) that separates a PWA from a regular web app are available through Service Workers [37]. A Service Worker is a JavaScript script which performs background operations separate from the rest of the website as it cannot interact with the DOM [38]. It acts as an application-level network proxy, allowing developers to fully intercept network requests and control such as caching of data and assets for offline availability, background synchronization (e.g. fetching data while app is in the background) and registering for push notifications [38,39]. As per July 2017, Apple's Safari browser does not support the Service Workers API, rendering their platform's users unable to use PWAs to their full extent.

Application Shell (AppShell). A well-designed and optimized *AppShell* is of utmost importance for an optimal user experience. This is the first-to-render user interface of the PWA, ensuring that the user experiences immediate rendering of *some content* to avoid the feeling of a slow app [40]. The AppShell is made up of static assets and interface components, meaning that it should not depend on any external dynamic data to render. Examples of such static assets include toolbars, navigation bars, splash screens and the like developed in HTML, CSS and JavaScript [40]. In contrast, dynamic data requires an asynchronous action to be fetched before render (e.g. network call or local database query).

Web App Manifest. The Manifest is a JSON file web developers can use to configure their PWAs. Within the file, configurations such as app name, icon paths, splash screen image, background colours and display types (e.g. fullscreen, with or without browser artefacts) can be specified [41]. Such configurations are important for enhancing the *app-like* feeling of a PWA, a concept further discussed in Sect. 4.3.

HTTPS. Progressive Web Apps must be served via HTTPS [38]. This ought to “[...] *prevent snooping and to ensure content hasn’t been tampered with.*”, according to Google’s LePage [42].

4.3 Concepts

A total of 10 concepts officially [42] lay the foundation of Progressive Web Apps. Some of these concepts are known from regular web development, while others are exciting additions to the mobile web.

Progressive. The *progressive* in Progressive Web Apps refers to the concept of *Progressive Enhancement*, which builds on the idea that a product, e.g. a website in our example, can progressively become better and more advanced based on the browser and device of a user. That means that users on devices with weaker hardware and fewer capabilities (such as lack of Service Worker support) can still visit and enjoy the experience of the website. Thus, Progressive Enhancement focusses on content over browser capabilities, i.e. functionality [43].

Responsive. At the heart of mobile-compliant web design is the concept of *responsive design* [44]. A responsive web design allows for different form factors to engage with the content of a website in an optimal fashion. This applies to the sizing of fonts, placement of images, stacking of grids, and so on [45]. This concept is not proprietary to PWAs, but rather essential for any website wishing to deliver a good user experience for visitors using mobile devices.

Installable. A Progressive Web App can be downloaded and installed onto devices directly from a user's Internet browser, as seen in Sect. 5.2 and Fig. 7. Together with the concepts discussed below, installable web apps greatly extends the possibilities of the mobile web. This is where the unification of traditional mobile apps and web apps starts. No longer are do apps have to be installed via constrained app stores, as they can now be installed through a PWA-compliant browser such as Chrome or Samsung Internet.

Connectivity Independent. This is one of the core concepts that differentiates a PWA from a regular web app or website. Where a regular website would require an active Internet connection to function and properly render content to the user, a PWA can manage offline contexts as well. It does this by leveraging Service Workers, as introduced in Sect. 4.2 on the underlying technologies of PWAs, as well as local data and asset caches [17].

App-Like. As Figs. 1, 2 and 5 illustrate, differentiating a Progressive Web App (Fig. 5) from regular apps (Figs. 1 and 2) is hard, if not impossible by judging only by the looks. A well-built and well-designed PWA can act, look and feel like a regular native or cross-platform app. Archibald [39] presented Google's vision for the mobile web with the following statement: "We want everything that ends up on the home screen to be competitive with native apps. We want to make the web a first-class part of the operating system in the user's mind".

Fresh. With the help of Service Workers, a PWA can fetch new content from an associated web service whenever the app requires it, either during a background synchronization task or whenever the user opens the app. This ensures that the app is both available offline, though with potentially dated cached content, and online with *fresh* content ready for digesting and caching for future offline use.

Safe. As discussed in Sect. 4.2, PWAs must be served via HTTPS for optimal end-user security.

Discoverable. Search engines can discover and index the content of Progressive Web Apps as they are indeed regular websites, only with additional capabilities as discussed in the rest of this section.

Re-engageable. Using the new Web Push API available in certain browsers [46], Progressive Web Apps can deliver the same *re-engaging* experience as regular apps by leveraging push notifications to draw users back into the app.

Linkable. Lastly, being *linkable* is part of any reachable website, PWA or not. Thus, PWAs should adhere to the same standard by providing easy sharing of

content through URLs. In fact, a PWA will also be able to register *intents*, for now on Android Chrome [47]. This allows an installed PWA to automatically open itself upon visiting or clicking a link related to the PWA website, e.g. open the reddit Art PWA automatically if visiting <https://www.reddit.com/r/Art/>.

4.4 Lighthouse: PWA Testing Tool

While not being a technology or concept, the Lighthouse Chrome extension is an official Google product targeting PWA developers. The Lighthouse extension will aid in benchmarking websites against certain measurements and aims Google assess as important, especially for PWAs. While the extension works on non-PWA websites too, it is designed to help optimize websites for better rendering speed, time to first possible user interaction and general compliance with Google’s aim for PWAs and the future of (mobile) web experience [48]. Lighthouse will generate a verbose report on the state of the tested website, provide resources on how to optimize code and assets, and give an overall score, as seen in Fig. 6 below.

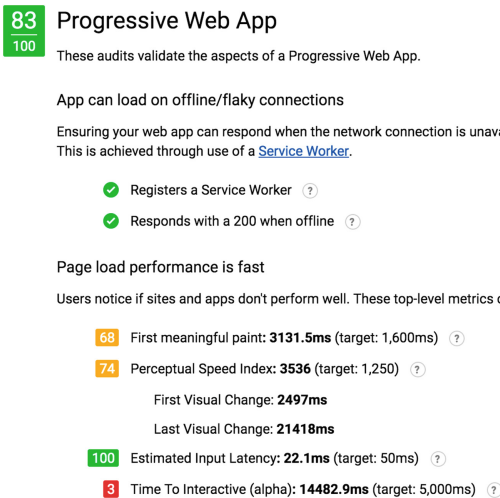


Fig. 6. Example of a Lighthouse report, generated using the reddit Art PWA.

4.5 Framework Agnosticity

An open-source initiative named HNPWA.com, short for *HackerNews PWA*, aims to aid developers in choosing between the vast amount of front-end web frameworks [49] available for building Progressive Web Apps.

As PWAs can be developed using any web framework or library [50], such an initiative is of utmost importance. HNPWA acts as a vehicle for increasing

popularity and clarity around what a Progressive Web App is – and what it can achieve in the context of user experience.

In Subsect. 4.4, the Lighthouse performance testing tool was introduced. This tool acts as the basis for the framework comparisons presented at HNPWA. Results of rigorous performance testing for a number of front-end frameworks and technologies are presented on HNPWA. This data can – and perhaps *should* – be used in decision making on technology choices, especially if PWA concepts are important for a project.

5 Results

In this section we present both technical and more overarching results from our study. The data presented should inspire academia to continue researching Progressive Web Apps and cross-platform development in general.

5.1 Feature Comparison

This section seeks to provide insights into differences between interpreted apps, Progressive Web Apps, hybrid apps and native apps by comparing a set of features and concepts. It also presents insights on such as technical frameworks and experience unification for end-users.

Table 1 provides a non-exhaustive list of features available in PWAs as of July 2017, along with their compatibility. Remarks follow subsequently.

Table 1. Feature-comparison of approaches.

Feature	Interpreted	PWA	Hybrid	Cross-compiled	Native
Installable	Yes	Yes ^a	Yes	Yes	Yes
Offline capable	Yes	Yes	Yes	Yes	Yes
Testable before installation	No	Yes	No	No	No
App marketplace availability	Yes	Yes ^b	Yes	Yes	Yes
Push notifications	Yes	Yes ^c	Yes	Yes	Yes
Cross-platform availability	Yes	Limited ^d	Yes	Yes	No
Hardware and Platform API access	Yes	Limited ^e	Yes ^f	Yes	Yes
Background synchronisation	Yes	Yes	Yes	Yes	Yes

- (a) The `Enable improved add to Home screen` developer flag in Chrome Canary for Android can be enabled in order for PWAs to be installed like normal apps [51]. This is as true as of July 2017 as it was in January 2017 [1].
- (b) PWAs will be made searchable from the Windows 10 app marketplace, thus becoming “first-class citizens” of their app ecosystem [52].
- (c) Push notification support through the `Push API` [53] is available, but limited to certain browsers.
- (d) As Apple’s Safari browser does not yet support the Service Workers API, the iOS platform is yet to fully realise and leverage the potential of the technological advancement.
- (e) A PWA can use HTML5-based APIs for hardware and platform access in addition to features and functionality made possible by Service Workers.
- (f) Hardware and platform access for Hybrid apps is usually provided by Cordova, a library for handling the bridging between a native app’s web-view component and the device’s APIs.

5.2 Unification of Mobile App and Web Experiences

An evident difference between web apps and regular mobile apps is their *explorability*. A regular app requires search and installation via an app marketplace. Progressive Web Apps enable the best of both approaches, where end-users can easily experience an application through their web browser, then choose to install it via an “Add to Home screen” banner prompted (see Fig. 7). The banner will be prompted to the user if certain qualifications are met [54], or can be displayed programmatically as controlled by the website’s developer [47]. Figure 7 illustrates the installation process for the PWA designed and implemented for this paper (cropped screenshots).

The illustrated experience is per July 2017 only achievable by enabling the `Improved PWA Installation` flag in the Chrome Canary for Android settings. Without that flag enabled, the experience is comparable to bookmarking a site to the home screen.

Together with the full screen experience of PWAs, the installation prompt can be considered an advance in unification of end-user experience and mobile web perception. Instead of forcing users to download an app from a marketplace, they can experience the product in their web browser as a regular website before installing it via the banner. As we discuss in our list of suggestions for future work, research on potential security concerns should be of interest to all parties.

5.3 Unification of Desktop App and Web Experiences

Developing cross-platform desktop applications using JavaScript frameworks such as Electron.js has over the last year seen an immense increase in popularity [55]. Mainstream software such as GitHub Desktop, Visual Studio Code

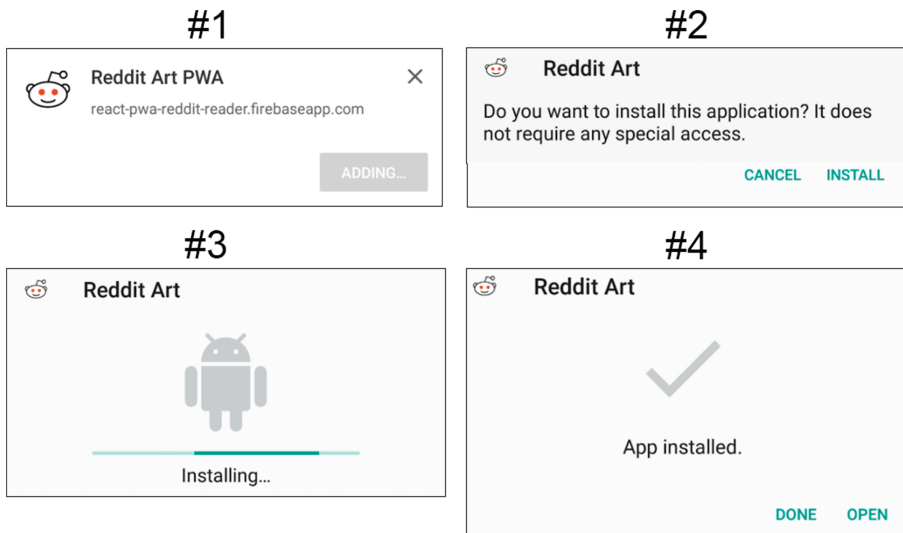


Fig. 7. PWA installation flow in Chrome Canary.

and Slack is now being written cross-platform using Electron.js [56]. As we introduce in this section, PWAs can perhaps be the future approach for developing such applications, without the need of Electron.js or similar frameworks.

A PWA's reach and deployability is not limited to mobile devices only. Both on Chrome OS, Windows and MacOS (previously OS X), certain browsers such as Google Chrome have started to bridge the gap between desktop apps and web apps by offering desktop PWA support [47,57]. Figure 8 depicts how a PWA in Chrome on MacOS prompts the user with an installation banner, asking them to "Add [the] site to [user's] shelf to use it any time", as of July 2017. This is the desktop version of the PWA banner seen in image #1 in Fig. 7.

Upon clicking the *Add* button in above banner, a dialog box is prompted to the user, as seen in Fig. 9. This action triggers the PWA to be downloaded to the user's computer and added to the *Chrome Apps* folder, on MacOS found at the following path: `/Users/<username>/Applications/Chrome Apps.localized`.

The downloaded version can now be executed at the user's machine and used in offline situations in the same way the PWA works on a mobile device. However, the user experience differs between desktop and mobile PWAs as of July 2017. Where a mobile PWA can execute in its own browser-artefact-less shell, i.e. looking like a regular mobile app, opening a desktop PWA will simply launch the downloaded app in the regular Chrome browser as a new tab. This is a major difference between a desktop PWA and an Electron.js desktop app. In Sect. 7.2 we propose topics and suggestions for further research. Continuing scrutinizing PWAs as a mean of cross-platform desktop development should be of profound interest for industry and academia alike.

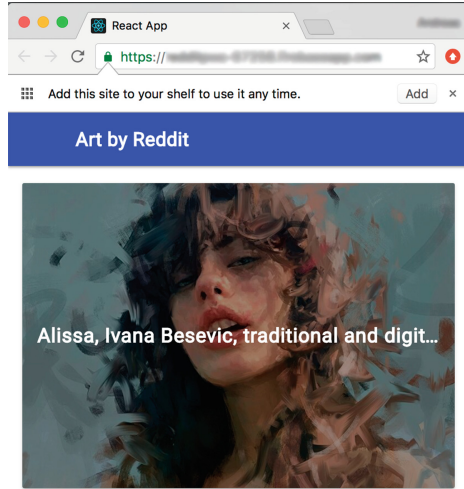


Fig. 8. The Progressive Web App running in Chrome on desktop.

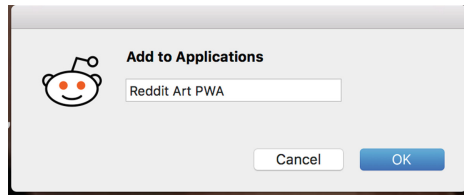


Fig. 9. Chrome prompting user with a dialogue after clicking the “Add” button.

5.4 Performance Comparison Between Native, Hybrid, Interpreted, Cross-Compiled and PWA

Table 2 presents a comparison of three different measurements: installation size, launch time and toolbar render time. The installation size of the Progressive Web App (104KB) is

- (a) 242 times smaller than the Xamarin-based cross-compiled app (25.19 MB),
- (b) 157 times smaller than the React Native-based interpreted app (16.39 MB),
- (c) 43 times smaller than the Ionic Framework-based hybrid app (4.53 MB), and
- (d) 42 times smaller than the native Android app (4.37 MB).

In terms of render-speeds, the PWA rendered different results when Chrome Canary (a) did not run in the background, and (b) it did run in the background, i.e. has previously been used to browse some arbitrary website.

Note that in our prior paper [1], the Ionic-based Hybrid app’s *Time from app-icon tap to toolbar render* speed clocked in at 9242.1 ms. We found that by

Table 2. Measurement-comparison of approaches.

Measure	Native	Hybrid	Interpreted	Cross-compiled	PWA
Size of installation	4.37 MB	4.53 MB	16.39 MB	25.19 MB	104 KB
Android activity launch time	1408 ms	467.5 ms	246 ms	4190 ms	230 ms
Time from app-icon tap to toolbar render	1688 ms	3999 ms	1716 ms	4590 ms	(a) 3152 ms (b) 1319 ms

re-implementing the app in a newer version of Ionic, as well as investigating a configuration file which contained an unnecessary splash screen delay, we got the render time down to 3999 ms, and Android activity launch time dropped from 860 ms to 467.5 ms. Without said investigation, the default delay would cause an additional 3000 ms to the render time [58].

It is important to note that the *activity launch* and *toolbar render* times do not stack, i.e. it does *not* e.g. take a total of 1408 ms + 1688 ms to render the native app’s toolbar. The native app, in this example, will launch the Android activity after 1408 ms after app icon tap (launch), then within the next 280 ms (1688 ms – 1408 ms) render the toolbar.

6 Discussion

Based on our results and findings, we use this section to scrutinize and further discuss the possibilities and limitations of Progressive Web Apps when compared to other cross-platform approaches.

6.1 Basics

From the perspective of web-native unification, there are certain differences and major limitations to the PWA approach compared to hybrid, interpreted, cross-compiled and native apps. We use this section to stimulate interest by elaborating on a set of these differences and limitations based on our findings, and suggest for future research to dive deeper.

As discussed by Malavolta [59], a PWA cannot access hardware- and platform-level features not supported by the respective browser. Examples of such non-included features are a native device calendar and contact list access. However, an increasing pool of platform APIs are becoming available in newer browsers [18].

For Progressive Web Apps to have the same potential as cross-platform app development, support for **Service Workers** in Apple’s iOS Safari browser is a requirement. Without such support, a PWA-enabled website would not deliver a consistent experience across browsers and platforms.

6.2 Feature Comparison

Table 1 highlighted differences in feature compatibility between approaches for app development. There are certain profound differences between them, some being inherent characteristics. In general, the interpreted and cross-compiled approaches provided the highest level of feature compliance.

A Progressive Web App is the only option among the listed approaches that naturally enables testing of an app *prior* to installation, due to being accessible in web browsers. If app marketplace presence is required, the four other approaches fully supports such, with a potential PWA entry into the Microsoft app store [52].

Offline capabilities, push notifications and background synchronisation are available regardless of approach. Cross-platform compatibility is found in the hybrid, interpreted, cross-compiled and PWA approaches, the latter with some limitations in regards to supported operating systems (no iOS support) and platform feature API access limited by browser device API implementations [60].

For apps relying on features not found in – or requiring performance not achievable in a modern-day browser, the alternative approaches are more suitable. The five artefacts developed as part of this paper are limited in terms of device-platform communication requirements, as they do not require any APIs other than those commonly found in browsers.

6.3 Performance Comparison

Table 2 presented the results from three measurements conducted for preliminary data gathering. It thereby also provides an anchor for further research.

The results illustrate the presence of certain trade-offs. If minimal app size is a priority, no approach is even remotely close to the 104 KB PWA. The second smallest in size is the *native* app, still weighing 42 times more (4.37 MB) than the PWA albeit rendering the toolbar faster in certain situations (with Chrome closed). The largest *and* slowest of all approaches was the cross-compiled Xamarin app. At 242 times (25.19 MB) the size of the PWA and over three times as slow (compared to launching the PWA with Chrome open), the cross-compiled approach overall performed worst in the test. It is interesting that the cross-compiled app performed this way, as the codebase has been compiled from a common language into native binaries, thus exposing a native user interface.

Yet, the hybrid approach, which, as discussed, is questionable when it comes to performance, managed to outperform the cross-compiled app in all three measurements. The hybrid approach also outperformed the native approach at activity launch (so does interpreted and PWA), but failed to deliver the same fast rendering speeds (native at 1688 ms, hybrid at 3999 ms).

While the interpreted app installation size was 157 times larger than the PWA, it was also more than 3.5 times faster to render the toolbar (first render) when launching the PWA without having Chrome running in the background. If launched with Chrome already running, the gap between the two apps was down to 457 ms, still favouring the interpreted app. If installation size is not the main priority of a project, the interpreted approach is seemingly producing

fast-launching, fast-rendering apps while still weighing 1.5 times less than the cross-compiled app. The interpreted and cross-compiled approaches are the only cross-platform approaches that would render native user interfaces (see Table 2). If cross-platform deployability and native interface performance are important criteria of a project, the interpreted approach would, based on our findings, be the one to adopt.

Interestingly, the rather grim assessment of the cross-platform approach seemingly contradicts earlier findings (cf. e.g. [2,61]). However, such work usually reports good results from technologically very sound approaches, such as from those based on model-driven techniques – which we explicitly excluded in this paper since such approaches do not play any practical role in the app development landscape of today.

The presented results are limited in that the apps were only tested on one specific device, the Google Nexus 5X, running Android 7.0. The conducted tests do not build on previous research or established methods for measuring performance. The purpose of the tests was to gather and present preliminary results to spark interest for further work. The differences may be more significant on older, especially on less powerful devices. Also, differences between technical frameworks within an approach and the apps they produce is likely to occur. The fact that Xamarin of the cross-compiled approach generated such subpar results compared to the other approaches does not inherently mean that all cross-compiled frameworks will do the same. We aim to conduct further research on PWAs and cross-platform development involving an array of devices and frameworks for increased validity. We deem our findings presented in Table 2 to be essential for decision making; we will continue to further explore possible measurements and technical approaches and frameworks, as well as to bolster the theory behind work such as the one presented here.

7 Conclusion and Further Work

We have reported on numerous findings and gaps throughout the paper. This section concludes our research and discussion, and provide suggestions for further research in hope of inspiring fellow academics to continue scrutinizing new possibilities with the introduction of Progressive Web Apps.

7.1 Conclusion

The Progressive Web App approach currently waits for – or probably even depends on – Apple implementing Service Workers into their iOS Safari browser. The approach could then reach its full potential.

With fast user interface rendering times (in some situations *even faster* than native) [39], overall well-compliant platform- and device API integrations via HTML5 and JavaScript, and the insignificant space they occupy on-device, PWA should be considered a very possible upcoming contender in the cross-platform space.

When compared to (other) cross-platform approaches, PWA lacks native user interfaces cross-compiled and interpreted frameworks will produce. However, so does the popular hybrid approach. Access to device and platform feature APIs is limited in PWAs to which such APIs are implemented into the users' browser. No such limitations have been identified for the other cross-platform approaches. However, due to being web-based PWAs can inherently be tested in any web browser before an eventual on-device installation in a Service Worker-compliant browser. This is unique to PWAs when compared to the other approaches. Service Workers enable PWAs to perform tasks such as background synchronization (allowing offline availability) and push notification registration. Such features were previously only available in native apps and via cross-platform approaches.

Progressive Web Apps do also have the potential of becoming an approach for developing desktop apps, as discussed in Subsect. 5.3. With Microsoft's future plans of including PWAs in their Windows Store, Google's plans of leveraging them on their ChromeOS platform, and the current possibilities of installing PWAs on desktop machines via e.g. the Chrome browser, we may see a shift in how desktop apps are developed. Through further research and industry adoption, desktop Progressive Web App ought to be further scrutinized.

We have identified a widening gap in the scientific literature concerning PWA. As of July 2017, less than a handful of papers form the entire published academic body of knowledge on the subject to the best of our knowledge. This makes for interesting research possibilities; we present suggestions in the following.

7.2 Suggestions for Further Work

We encourage academia to keep track of the progress in the field of mobile computing and the web, as well as their intersection. Both technical and non-technical possibilities emerge with the introduction of Progressive Web Apps across devices and platforms.

Based on our study and understanding of the field, we suggest the following topics for further research:

- Expand the testing and assessment activities, including additional measurements e.g. regarding energy consumption.
- Scrutinize PWAs' capabilities on desktop compared to cross-platform desktop frameworks such as Electron.js and traditional development approaches.
- Investigate if – and possibly how – PWAs could be supported on other devices, such as smart TVs (e.g. Samsung Tizen) and Wearables such as smart watches.
- Research patterns of technical development and of application for optimizing web applications for an offline-first paradigm shift.
- Further investigate, develop, compare and evaluate technical frameworks for Progressive Web App – and cross-platform development.
- Evaluate embeddable databases for (Progressive) Web Apps in the context of offline use.

- As the PWA installation banner can now be programmatically handled by developers [47], study how and when users should be prompted with the banner for an optimal number of installation.
- Study Progressive Web Apps' effect on user retention compared to regular mobile apps (both native and cross-platform). An hypothesis could be that the insignificant size of a PWA on a space-constrained mobile device leads to less app being uninstalled compared to the usage of regular apps.
- As arbitrary (PWA) code and associated assets now can be downloaded to various devices, are there any new security concerns introduced? In fact, the Android world recently witnessed how a game developer managed to nearly brick a customer's phone with their Android Play Store game [62], breaking out of the *Android application sandbox*. Thus, proper assessment of possible newly introduced security risks should be prioritized.
- Include business users and consumers in studies, for example regarding the user perception of PWAs.

References

1. Biørn-Hansen, A., Majchrzak, T.A., Grønli, T.M.: Progressive web apps: the possible web-native unifier for mobile development. In: Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST), pp. 344–351. SciTePress (2017)
2. Heitkötter, H., Majchrzak, T.A., Kuchen, H.: Cross-Platform Model-Driven Development of Mobile Applications with MD². In: Proceedings of 28th ACM SAC, pp. 526–533. ACM (2013)
3. Perchat, J., Desertot, M., Lecomte, S.: Component based framework to create mobile cross-platform applications. In: Procedia Computer Science, vol. 19, pp. 1004–1011. ScienceDirect (2013). <http://www.sciencedirect.com/science/article/pii/S1877050913007485>
4. Majchrzak, T.A., Heitkötter, H.: Status quo and best practices of app development in regional companies. In: Krempels, K.-H., Stocker, A. (eds.) WEBIST 2013. LNBIP, vol. 189, pp. 189–206. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44300-2_12
5. Malavolta, I., Ruberto, S., Soru, T., Terragni, V.: Hybrid mobile apps in the Google play store: an exploratory investigation. In: Proceedings 2nd ACM International Conference on Mobile Software Engineering and Systems, MOBILESoft 2015, pp. 56–59. IEEE Press (2015)
6. Angulo, E., Ferre, X.: A case study on Cross-Platform development frameworks for mobile applications and UX. In: Proceedings of the XV International Conference on Human Computer Interaction, p. 27. ACM (2014)
7. Heitkötter, H., Hanschke, S., Majchrzak, T.A.: Evaluating cross-platform development approaches for mobile applications. In: Cordeiro, J., Krempels, K.-H. (eds.) WEBIST 2012. LNBIP, vol. 140, pp. 120–138. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36608-6_8
8. Majchrzak, T.A., Biørn-Hansen, A., Grønli, T.M.: Comprehensive analysis of innovative cross-platform app development frameworks. In: Proceedings of 49th HICSS. IEEE Computer Society (2017)

9. Rahul, R., Tolety, S.B.: A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In: Annual IEEE India Conference, pp. 625–629. IEEE (2012). <https://doi.org/10.1109/INDCON.2012.6420693>
10. El-Kassas, W.S., Abdullah, B.A., Yousef, A.H., Wahba, A.M.: Taxonomy of Cross-Platform mobile applications development approaches. *Ain Shams Eng. J.* **8**, 163–190 (2017)
11. Rieger, C., Majchrzak, T.A.: Weighted evaluation framework for cross-platform app development approaches. In: Wrycza, S. (ed.) SIGSAND/PLAIS 2016. LNBIP, vol. 264, pp. 18–39. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46642-2_2
12. Du, F.Q.: Under the Hood: Rebuilding Facebook for Android (2012). <https://www.facebook.com/notes/facebook-engineering/under-the-hood-rebuilding-facebook-for-android/10151189598933920/>. Accessed 13 Jan 2017
13. Lee, C.: Zuckerberg talks about Facebook App mistakes, rumored smartphone and more (2012). <http://www.idownloadblog.com/2012/09/11/zuckerberg-app-phone-more/>. Accessed 17 Apr 2016
14. Latif, M., Lakhri, Y., Nfaoui, E.H., Es-Sbai, N.: Cross platform approach for mobile application development: A survey. In: 2016 International Conference on Information Technology for Organizations Development (IT4OD), pp. 1–5. IEEE (2016)
15. Ciman, M., Gaggi, O.: An empirical analysis of energy consumption of cross-platform frameworks for mobile development. *Pervasive Mobile Comput.* **39**, 214–230 (2016). <https://doi.org/10.1016/j.pmcj.2016.10.004>
16. Russel, A., Berriman, F.: Progressive web apps: escaping tabs without losing our soul (2015). <https://goo.gl/e6pZHF>
17. Osmani, A., Cohen, M.: Offline storage for progressive web apps (2017). <https://developers.google.com/web/fundamentals/inline-and-offline/web-storage/offline-for-pwa>. Accessed 3 July 2017
18. Roy-Chowdhury, R.: The mobile web: state of the union (2017). https://www.youtube.com/watch?v=_ssDaecATCM
19. Malavolta, I., Procaccianti, G., Noorland, P., Vukmirović, P.: Assessing the impact of service workers on the energy efficiency of progressive web apps. In: Proceedings of the 4th International Conference on Mobile Software Engineering and Systems, MOBILESoft 2017, Piscataway, NJ, USA, pp. 35–45. IEEE Press (2017). <http://dl.acm.org/citation.cfm?id=3104093>
20. Gudla, S.K., Sahoo, J.K., Singh, A., Bose, J., Ahamed, N.: Framework to improve the web application launch time. In: Proceedings of 2016 IEEE International Conference on Mobile Services (MS), pp. 73–78. IEEE Press (2016)
21. Fransson, R., Driaguine, A.: Comparing progressive web applications with native android applications: an evaluation of performance when it comes to response time. Ph.D. thesis, Linnaeus University (2017). <http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1105475&dswid=-7607>
22. Shafirov, M.: Kotlin on Android. Now official (2017). <https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/>. Accessed 9 June 2017
23. Majchrzak, T.A., Ernsting, J., Kuchen, H.: Achieving business practicability of model-driven cross-platform apps. *Open J. Inf. Syst.* (OJIS) **2**, 3–14 (2015)
24. Android Developers: (Implementing descendant navigation). <https://developer.android.com/training/implementing-navigation/descendant.html>. Accessed 12 July 2017
25. Ater, T.: Building Progressive Web Apps: Bringing the Power of Native to the Browser. O’Reilly (2017)

26. Hume, D.A.: Progressive Web Apps. Manning (2017). <https://www.manning.com/books/progressive-web-apps>
27. Edwards, A.R.: The building blocks of progressive web apps - smashing magazine (2016). <https://www.smashingmagazine.com/2016/09/the-building-blocks-of-progressive-web-apps/>
28. Mahemoff, M.: Progressive web apps have leapfrogged the native install mode ... but challenges remain (2016). <https://goo.gl/EbnF0N>
29. Rinaldi, B., Holland, B., Looper, J., Motto, T., VanToll, T.J.: Are progressive web apps the future? (2016). <http://developer.telerik.com/featured/are-progressive-web-apps-future/>
30. Xanthopoulos, S., Xinogalos, S.: A comparative analysis of cross-platform development approaches for mobile applications. In: Proceedings of 6th Balkan Conference in Informatics, BCI 2013, pp. 213–220. ACM (2013)
31. Dalmasso, I., Datta, S.K., Bonnet, C., Nikaein, N.: Survey, comparison and evaluation of cross platform mobile application development tools. In: Proceedings of 9th IWCMC, pp. 323–328 (2013)
32. Palmieri, M., Singh, I., Cichetti, A.: Comparison of cross-platform mobile development tools. In: Proceedings of 16th International Conference on Intelligence in Next Generation Networks, pp. 179–186. IEEE (2012)
33. Gaouar, L., Benamar, A., Bendimerad, F.T.: Desirable requirements of cross platform mobile development tools. *Electron. Dev.* **5**, 14–22 (2016)
34. Mercado, I.T., Munaiah, N., Meneely, A.: The impact of cross-platform development approaches for mobile applications from the user's perspective. In: Proceedings of International Workshop on App Market Analytics, WAMA 2016, pp. 43–49. ACM (2016)
35. Malavolta, I., Ruberto, S., Soru, T., Terragni, V.: End users' perception of hybrid mobile apps in the Google play store. In: 2015 IEEE International Conference on Mobile Services, pp. 25–32. IEEE (2015)
36. Rieger, C., Majchrzak, T.A.: Conquering the mobile device jungle: towards a taxonomy for app-enabled devices. In: Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST), pp. 332–339. SciTePress (2017)
37. Service Workers 1 (2017). <https://developers.google.com/web/fundamentals/getting-started/primers/service-workers>. Accessed 29 June 2017
38. Gaunt, M.: Service workers: an introduction (2016)
39. Archibald, J.: Instant loading: building offline-first progressive web apps (2016). <https://www.youtube.com/watch?v=cmGr0RszHc8>
40. Osmani, A., Gaunt, M.: Instant loading web apps with an application shell architecture (2017). <https://developers.google.com/web/updates/2015/11/app-shell>
41. Guant, M., Kinlan, P.: The web app manifest (2016). <https://developers.google.com/web/fundamentals/engage-and-retain/web-app-manifest/>. Accessed 8 Dec 2016
42. LePage, P.: Your first progressive web app (2017). <https://developers.google.com/web/fundamentals/getting-started/codelabs/your-first-pwapp/>. Accessed 8 June 2017
43. Cazañas, A., Parra, E.: Strategies for mobile web design. In: INCISCOS 2016 (2017). ingenieria.ute.edu.ec
44. 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). https://doi.org/10.1007/978-3-642-39200-9_47

45. LePage, P.: Responsive web design basics (2017). <https://developers.google.com/web/fundamentals/design-and-ui/responsive/>. Accessed 4 July 2017
46. CanIUse: Can I use push API (2017). <http://caniuse.com/#feat=push-api>. Accessed 29 June 2017
47. Birch, S., Russell, A.: Progressive web apps: great experiences everywhere (2017). <https://www.youtube.com/watch?v=m-sCdS0sQO8>
48. Google Developers: Lighthouse (2017). <https://developers.google.com/web/tools/lighthouse/>. Accessed: 2017-6-28
49. Smeets, R., Aerts, K.: Trends in web based cross platform technologies. *Int. J. Comput. Sci. Mobile Comput.* **5**, 190–199 (2016)
50. Osmani, A.: Production progressive web apps with JavaScript frameworks (2017). <https://www.youtube.com/watch?v=aCMbSyngXB4>
51. Joreteg, H.: Installing web apps on phones (for real) (2016). <https://joreteg.com/blog/installing-web-apps-for-real>
52. Rossi, J.: The progress of web apps - microsoft edge dev blog (2016). <https://blogs.windows.com/msedgedev/2016/07/08/the-progress-of-web-apps/>
53. Push API (2017). <https://www.w3.org/TR/push-api/>. Accessed 29 June 2017
54. Pedersen, M.: Progressive web apps: Bridging the gap between web and mobile (2016). <https://www.sitepoint.com/progressive-web-apps-bridging-the-gap-between-web-and-mobile/>
55. npm-stat: npm-stat: electron (2017). <https://npm-stat.com/charts.html?package=electron&from=2015-11-01&to=2017-06-28>. Accessed 28 June 2017
56. Electron: (Electron). <https://electron.atom.io/>. Accessed 28 June 2017
57. Raible, M.: The ultimate guide to progressive web applications (2017). <http://scotch.io/tutorials/the-ultimate-guide-to-progressive-web-applications>. Accessed 28 June 2017
58. Ionic: ionic2-app-base (2017). <https://github.com/ionic-team/ionic2-app-base/blob/master/config.xml#L30>
59. Malavolta, I.: Beyond native apps: web technologies to the rescue! (keynote). *ACM* (2016). <http://doi.acm.org/10.1145/3001854.3001863>
60. Puder, A., Tillmann, N., Moskal, M.: Exposing native device APIs to web apps. In: *Proceedings of 1st International Conference on Mobile Software Engineering and Systems*, pp. 18–26. *ACM* (2014). <https://doi.org/10.1145/2593902.2593908>
61. Heitkötter, H., Majchrzak, T.A.: Cross-platform development of business apps with MD². In: vom Brocke, J., Hekkala, R., Ram, S., Rossi, M. (eds.) *DESRIST 2013*. LNCS, vol. 7939, pp. 405–411. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38827-9_29
62. Postfu: Is it actually possible to earn money from mobile game ads? (2017). https://np.reddit.com/r/gamedev/comments/6575nr/is_it_actually_possible_to_earn_money_from_mobile/dg8mmia/?context=10000. Accessed 28 June 2017



Web Performance Characteristics of HTTP/2 and Comparison to HTTP/1.1

Robin Marx^(✉), Maarten Wijnants^(✉), Peter Quax^(✉), Axel Faes^(✉),
and Wim Lamotte^(✉)

UHasselt-tUL-imec, EDM, Hasselt, Belgium
{robin.marx,maarten.wijnants,peter.quax,axel.faes,
wim.lamotte}@uhasselt.be

Abstract. The HTTP/1.1 protocol has long been a staple on the web, for both pages and apps. However, it has started to show its age, especially with regard to page load performance and the overhead it entails due to its use of multiple underlying connections. Its successor, the newly standardized HTTP/2, aims to improve the protocol's performance and reduce its overhead by (1) multiplexing multiple resources over a single TCP connection, (2) by using advanced prioritization strategies and by introducing new features such as (3) Server Push and (4) HPACK header compression.

This work provides an in-depth overview of these four HTTP/2 performance aspects, discussing both synthetic and realistic experiments, to determine the gains HTTP/2 can provide in comparison to HTTP/1.1 in various settings. We find that the single multiplexed connection can actually become a significant performance bottleneck in poor network conditions with high packet loss and that HTTP/2 rarely improves much on HTTP/1.1, except in terms of reduced overhead. Prioritization strategies, Server Push and HPACK compression are found to have a relatively limited impact on web performance, but together with other observed HTTP/2 performance problems this could also be due to faulty current implementations, of which we have discovered various examples.

Keywords: HTTP/2 · Web performance · Best practices · HTTP Server push · Prioritization · Networking · Measurements

1 Introduction

As the web grows more mature in terms of availability and features, so does its complexity. Websites have evolved from collections of simple individual document pages into complex user experiences and even full “apps”. Even though internet connection speeds have also been steadily increasing in this time frame, the traditional internet protocols HTTP/1.1 and TCP have struggled to keep up with these developments and are in many cases unable to provide fast web

page load performance [12]. This is detrimental to the overall viability of the web platform for complex use cases such as e-commerce, since a multitude of studies has shown that web performance is a core tenet in ensuring user satisfaction [9, 13].

Most of the performance problems with HTTP/1.1 stem from the fundamental limitation to only request a single resource per underlying TCP connection at the same time. This means that slow or large resources can delay others, which is called “Head-Of-Line (HOL) blocking”. As modern websites consist of tens to even hundreds of individual resources, browsers typically open several parallel HTTP (and thus also TCP) connections (up to six per hostname and 30 in total in most browser implementations). However, these heavily parallelized setups induce large additional overheads (e.g., in terms of server-side connection count) while not providing extensive performance benefits in the face of ever more complex websites [12] (see also Sect. 4). Note that while the HTTP/1.1 specification does include the *pipelining* technique (which does allow multiple requests to be queued on a connection), it is not enabled by default in the major modern browsers due to various practical issues [14].

In order to tackle these challenges, the new HTTP/2 protocol [3] (h2) was standardized in 2016, after evolving from Google’s SPDY protocol since 2009. While keeping full backwards compatibility with the semantics of the HTTP/1.1 protocol (h1) (e.g., types of headers, verbs and overall setup), h2 nevertheless introduces many low-level changes, primarily with the goal of improving web page load performance. For example, all h2 traffic is ideally sent over a single TCP connection (making use of multiplexing and inter-resource prioritization algorithms to eliminate HOL blocking), there is support for server-initiated traffic (Server Push) and headers are heavily compressed in the HPACK format. The details of these aspects are discussed in Sects. 4 to 7.

In theory, h2 should solve most of the problems of h1 and improve web page load times by up to 50% [12]. In practice however, the gains from h2 are limited by other factors and implementation details. Firstly, the use of a single TCP connection introduces a potential single-point-of-failure when high packet loss is present (and so it might actually be better to also use multiple TCP connections for h2). Secondly, correctly multiplexing multiple resources over this connection is heavily dependent on the used resource prioritization scheme and the interleaving of resource chunks might introduce its own implementation overhead as these chunks need to be aggregated before processing. Finally, complex interdependencies between resources and late resource discovery might also lessen the gains from h2 [22]. The fact that h2 is not a simple drop-in replacement with consistently better performance than h1 is also clear from previous studies, which often find cases where h2 is significantly slower than h1 (see Sect. 2).

In this text, we continue the groundwork from our previous publications [17, 18]. We discuss four HTTP/2 performance-related aspects and test their impact, both in synthetic and realistic test scenarios, in comparison with HTTP/1.1’s performance (Sects. 4 to 8).

Our main contributions are as follows:

- We extend the **Speeder framework for web performance measurement** [18], combining a large number of off-the-shelf software packages to provide various test setup permutations, leading to a broad basis for comparison and interpretation of experimental results.
- We compare **h2** to **h1** in both synthetic and realistic experiments and find that **while h2 rarely significantly improves performance over h1, it is also rarely much slower**. Additionally, in most cases, bad network conditions do not seem to impact **h2** much more than they impact **h1**. Using multiple parallel TCP connections can help both **h2** and **h1**. Prioritization, Server Push and HPACK compression seem to contribute only sparingly to page load time improvements.
- We find that many current **h2 implementations (both on the server and browser sides) are not yet fully mature** and that some (default) implementations lead to suboptimal performance, especially concerning the time it takes to start rendering the web page.

2 Related Work

Various authors have published works comparing the performance of **h2** and its predecessor SPDY to **h1**.

In “How Speedy is SPDY?” [27] the authors employ isolated test cases to better assess the impact of various parameters (latency, throughput, loss rate, initial TCP window, number of objects and object sizes). They observe that SPDY incurs performance penalties when packet loss is high (mainly due to the single underlying TCP connection) but helps for many small objects, as well as for many large objects when the network is fast. For real pages, they find that SPDY improves page load performance for up to 80% of pages under low throughput conditions, but only 55% of pages under high bandwidth.

“Towards a SPDY’ier Mobile Web?” [8] performs an analysis of SPDY over a variety of real networks and finds that underlying cellular protocols can have a profound impact on its performance. For 3G, SPDY performed on a par with **h1**, with LTE showing slight improvements over **h1**. A faster 802.11g network did yield improvements of 4% to 56%. They further conclude that using multiple concurrent TCP connections does not help SPDY.

“Is The Web HTTP/2 Yet?” [26] measures page load performance by loading real websites over real networks from their original origin servers. They find that most websites distribute their resources over multiple backend hosts and as such use **h2** over multiple concurrent connections, which “makes **h2** more resilient to packet loss and jitter”. They conclude that 80% of the observed pages perform better over **h2** than over **h1** and that **h2**’s advantage grows in mobile networks. The remaining 20% of the pages suffer a loss of performance.

“HTTP/2 Performance in Cellular Networks” [10] introduces a novel network emulation technique based on measurements from real cellular networks. They use this technique to specifically assess the performance impact of using multiple

concurrent TCP connections for **h2**. They find that **h2** performs well for pages with large amounts of small and medium sized objects, but suffers from higher packet loss and larger file sizes. They demonstrate that **h2** performance can be improved by using multiple connections, though it will not always reach parity with **h1**.

“HTTP/1.1 Pipelining vs HTTP2 In-The-Clear: Performance Comparison” [7] compares the cleartext (non-secure) versions of **h1** and **h2** (**h1c** and **h2c** respectively) (even though **h2c** is not currently supported by any of the main browsers, see Sect. 3). They disregard browser computational overhead and find that on average **h2c** is 15% faster than **h1c** and “**h2c** is more resilient to packet loss than **h1c**”.

Additional academic work [15] found that for a packet loss of 2%, “**h2** is completely defeated by **h1**” and that even naive Server Push schemes can yield up to 26% improvements. Others [25] conclude that **h2** is mostly interesting for websites with large amounts of images, showing up to a 48% decrease in page load time, with an additional 10% when using **h2** Server Push, and that **h2** is resilient to higher latencies but not to packet loss. Further experiments [23] indicate that **h2** Server Push seems to improve page load times under almost all circumstances. Finally, Carlucci et al. [6] state that packet loss has a very high impact on SPDY, amounting to a 120% increase of page load time compared to **h1** on a high bandwidth network.

Content Delivery Network (CDN) companies have also measured **h2** performance on their networks. Gooding et al. [11] from Akamai find that using multiple TCP connections is best avoided for critical resources on **h2**. A presentation by Fastly [2] states that **h2** mostly outperforms **h1** on fast networks, but loses on networks with higher packet loss.

Our review of related work clearly shows that the current state of the art is often contradictory in its conclusions regarding **h2** performance. It is not clear whether using multiple TCP connections provides significant benefits, whether **h2** is resilient to poor network conditions and what degrees of improvement developers might expect when migrating from **h1** to **h2**.

In this work, we try to assess why these contradictions exist by running a wide variety of tests on several heterogeneous test setups (see Sect. 3). We look at four performance-related aspects of **h2**, first in isolation to assess their relative impacts and then in combination to evaluate the protocol’s impact on typical realistic web page loads. We are thus able to confirm some of the findings reported by the related work, while showing that many of the contradictory findings can be attributed to inefficiencies in current **h2** implementations.

3 Experimental Setup with the Speeder Framework

As discussed in Sect. 2, there are many cases of contradictory results regarding the performance of **h2**. As we suspect that one of the reasons for these discrepancies are differences in the underlying **h2** implementations (both client/browser-side and server-side) and utilized test configurations, we aim to employ as many

test setup permutations as possible. We argue that if the results show similar trends across all or a large part of the test setups, they are most likely attributable to the protocol itself. If however the results vary widely, they are typically dependent on specific implementations.

In order to obtain these diverse test setup permutations, we use the Speeder framework for web performance measurement, previously introduced in [18]. Speeder provides pre-installed versions of a large amount of existing software packages (e.g., servers, browsers, network emulation tools, automated testing tools) that can be freely coupled to each other through the use of Docker containers¹. Users simply need to select the desired setup permutations and the framework collects and aggregates a multitude of key metrics. Users can then utilize various visualization tools to compare the results.

For this work, we have expanded Speeder in a variety of ways. We have upgraded the supported browser versions of Chrome and Firefox to v60 and v54 respectively, updated webpagetest² to v3 and now also support the H2O webserver³, which was heavily optimized for h2 from the ground up. We have also created and integrated the H2Vis visualization tool. H2Vis directly takes the low-level *.pcap* packet capture files recorded during a test run (using *tcpdump*⁴) to produce a number of insightful graphical representations. For example, we can plot both TCP-level and h2-level packets on a graphical timeline to help verify how data is actually sent by the h2 server, how the various h2 streams are interleaved on a single TCP connection (see Sects. 4 and 5) and what the practical impact of packet loss is on the connection. Additionally, support for graphically visualizing the generated h2 priority dependency trees (see Sect. 5) allows us to quickly assess the impact of various prioritization strategies in use by browsers. Table 1 provides an overview of the features of the Speeder framework at the time of writing.

Unless indicated otherwise, the results in this work were generated in an experimental setup using NGINX v1.10 as web server and Google Chrome v54 as browser, driven by Webpagetest v2.19 and the dynamic cellular network model. This dynamic network model uses previous work [10] which introduced a model based on real-life cellular network observations. The model has six levels of “user experience (UX)”: NoLoss, Good, Fair, Passable, Poor and VeryPoor. Each UX level contains a time series of values for bandwidth, latency and loss. The model changes these parameters at 70 ms intervals to simulate a real network. This implies, for example, that applied packet loss is more bursty than with the fixed model. For details, please see [10] or the original source code⁵.

Our results will be presented using two distinct metrics, namely `loadEventEnd` and `SpeedIndex`. `loadEventEnd` from the Navigation Timing API [28] gives a good indication of the total time (in milliseconds (ms)) a page needed

¹ <https://www.docker.com/>.

² <https://www.webpagetest.org/>.

³ <https://h2o.example.net/>.

⁴ <http://www.tcpdump.org/>.

⁵ <https://github.com/akamai/cell-emulation-util>.

Table 1. Software, metrics and visualizations supported in the Speeder framework (August 2017).

Protocols	HTTP/1.1 (cleartext), HTTPS/1.1, HTTPS/2
Browsers	Chrome (v51–v60), Firefox (v45–v54)
Test drivers	Sitespeed.io (v3), Webpagetest (v3.0)
Servers	Apache (v2.4.20), NGINX (v1.10), NodeJS (v6.2.1), H2O (v2.1)
Network	- DUMMYNET (cable and cellular) (provided by Webpagetest) - Fixed TC NETEM (cable and cellular) - Dynamic TC NETEM (cellular) [10]
Metrics	All Navigation Timing values [28], SpeedIndex [19], firstPaint, visualComplete, other Webpagetest metrics [20]
Visualizations	Packet timeline (TCP and h2), h2 priority dependency trees. Boxplots, linegraphs and CDFs of recorded metrics

to load, but does not say anything about how progressively it was rendered in that time frame. In other words: a page that stays empty for 5 s and only renders content during the last 0.6 s (page A) will have a better observed `loadEventEnd` performance than a page that finishes loading at 7.5 s, but that had its main content drawn by 2.5 s (page B), while the latter arguably yields the better end-user experience. In order to capture the degree to which the page loads progressively, Google introduced the `SpeedIndex` metric [19], which measures how fast a page renders, not just loads. Inconsistencies between `loadEventEnd` and `SpeedIndex` results can indicate that a resource was fast to load but slow to have visual impact. Like `loadEventEnd`, `SpeedIndex` is expressed in ms and so for both metrics lower values mean better performance.

Finally, we performed most of our tests using three versions of the HTTP protocol: the secure HTTPS/2 (`h2s`) and HTTPS/1.1 (`h1s`) and also the unencrypted HTTP/1.1 (`h1c`), because many websites still use this “cleartext” version. We do not include `h2c`, as modern browsers choose to only support `h2s` for security reasons. Note additionally that switching from `h1c` to a secure setup (either `h1s` or `h2s`) could have its own performance impact as TLS connections typically require additional network round-trips to setup. In the following sections, we will use `h2` to refer to `h2s`, and `h1` refers to both `h1s` and `h1c`.

Most of our graphs will show `loadEventEnd` on the Y-axis. Individual data points will typically represent aggregates (e.g., median, average) of 10 to 100 page loads. Each experiment was repeated at least five times. Unexpected datapoints and anomalies across runs were analyzed further by manually checking the collected output of individual page loads (e.g., screenshots/videos, .har files, waterfall charts, .pcap files). The line plots will show the median values under Good network conditions, as do the Cumulative Distribution Functions (CDFs). The box plots will show the median as a horizontal bar and the average as a black square dot, along with the 25th and 75th percentiles and min and max values as the whiskers. Some box plots use a logarithmic scale on the Y-axis to

allow for large values. To be able to compare our results using the SpeedIndex metric, we make sure our loaded resources have a strong visual impact on the visible “above the fold” part of the website.

Some of our results were obtained using hand-crafted experiments on synthetic data. These test cases are intended to demystify the underlying behavior of the protocols and their implementations, and so are often not entirely realistic or involve extreme circumstances. However, most of our results were obtained using more realistic data based on existing websites. We expect that, compared to the experiments on synthetic pages, these test cases will show similar but more nuanced results and trends.

Readers are encouraged to review our full dataset (which encompasses results not presented in this paper (e.g., for other browser/server combinations and test pages)), setup details and source code via <https://speeder.edm.uhasselt.be>.

4 Multiplexing over a Single TCP Connection

4.1 Background

One of the major downsides of HTTP/1.1 is that it only allows a single resource to be requested and sent on an individual TCP connection at any given time. As such, the problem of Head-Of-Line (HOL) blocking is introduced, where the delivery of the initial resource(s) can block later resources (e.g., if the initial resource is very slow to be generated, is very small (so it does not take up the full possible bandwidth) or is very large). To work around this problem, modern browsers typically open up to six parallel HTTP/TCP connections to a single origin server. This way, even if one or more of the connections suffer from HOL blocking, the others can serve key resources as soon as possible. In tandem, developers have adopted the practice of merging several smaller resources into larger files, a practice called “concatenation” or “bundling”. This approach causes the number of individual resources to go down and with them the number of needed TCP connections and HTTP requests. On the other hand, concatenation has the adverse effect that it reduces the fine-grained cacheability of individual, smaller resources.

Another **h1** best practice is that of “hostname sharding”. Web developers will typically distribute their resources over a number of individual servers with different hostnames (for example by using a CDN). The browser will open up to six connections per hostname, resulting in a total of 17–60 parallel **h1** connections across all hostnames⁶ per page load. This leads to massively parallel page loads, but also introduces significant overheads on the server side in order to support this large amount of connections and their state management. The downsides of both concatenation and sharding (reduced cacheability and higher overhead, respectively) do not always outweigh their observed page load performance benefits [12].

⁶ <http://www.browserscope.org/>.

In response, HTTP/2 tries to solve the root issue of HOL blocking by delivering multiple resources over a single TCP connection concurrently, using multiplexing. In practice, smaller chunks of individual resources are encapsulated in conceptual “streams” and are then interleaved on the single connection. Section 5 discusses in detail how **h2** decides on the resources’ interleaving order with a priority-based dependency tree. The HTTP/2 specification [3] actively encourages this single connection setup. For example, it includes a mechanism for coalescing requested HTTP connections to separate hostnames onto a single TCP connection if the hosts use the same HTTPS certificate and resolve to the same IP address, this way effectively “undoing” a typical sharded **h1** setup. HTTP/2 Server Push can also only be used for resources on the same domain (see Sect. 6).

In theory, **h2**’s approach should render the **h1** best practices of concatenation and sharding obsolete [12]. In practice however, the single TCP connection might also be more susceptible to adverse network conditions than **h1**’s parallel approach. With **h1**, if one or more of the parallel connections would incur packet loss or high jitter, the possibility exists that the other connections would remain unimpaired. With just a single **h2** connection, all resources will be impacted when the network deteriorates. In effect, this could introduce transport-layer HOL blocking, induced by TCP’s guarantee of in-order delivery combined with re-transmits when packet loss is present [21]. If the impact of packet loss is significant, **h2** might in practice also benefit from sharding on multiple connections (see Sect. 4.2).

4.2 Head-of-Line Blocking in Practice with Images

In order to assess the impact of concatenation and sharding on both **h1** and **h2** page load performance in varying network conditions, the experiments in Fig. 1 compare three cases: (left) concatenated into a single resource on one host, (middle) non-concatenated on one host, (right) non-concatenated on four hosts (“sharded”). In practice, for the sharded case, for **h1** the browser will open the maximum amount of connections (24, six per hostname) and a single connection per hostname for **h2** (four in our case). The observed **h1** connections are all configured with `Keep-Alive` and do not use pipelining.

In **h1** the problem of HOL blocking is most apparent when trying to download many smaller files, as browsers only open six parallel connections. Since these smaller files do not fully take up the available bandwidth and each individual resource request requires a full Round-Trip-Time (RTT) delay, this overhead quickly adds up. For this, we consider three experiments in Fig. 1: (a) a large number (i.e., 380) of small files, (b) a medium number (i.e., 42) of medium sized files and (c) a medium number (i.e., 30) of large files. We choose images because they typically incur a low processing overhead from the browser. We look at more complex JavaScript/CSS cases in the next section.

For Fig. 1(a) we observe that **h2** significantly outperforms **h1** when there is no concatenation (middle), but that using a single concatenated image largely reduces **h2**’s benefit and brings it somewhat on a par with **h1** (left). This is expected as the single **h2** connection can efficiently multiplex the many small

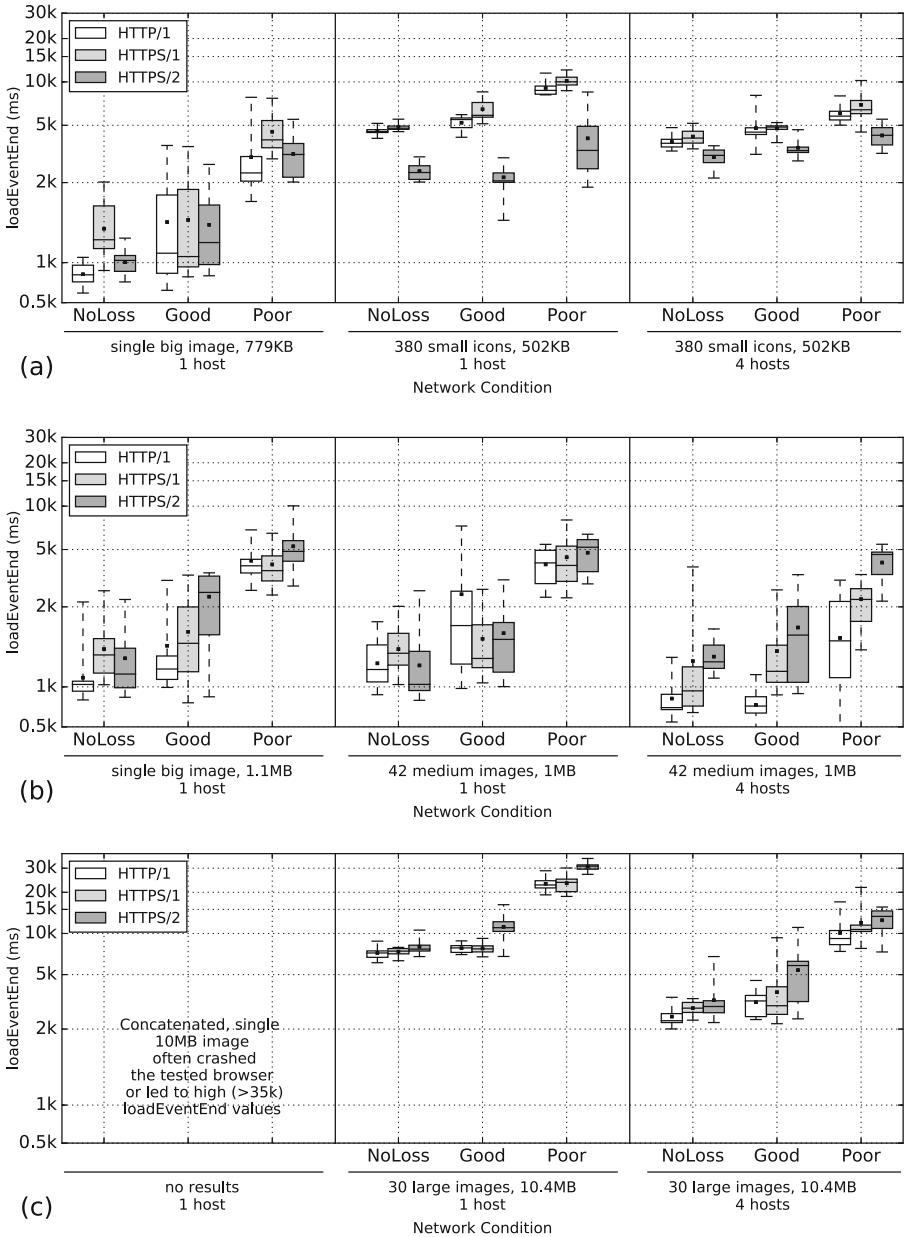


Fig. 1. Synthetic test cases concerning HOL blocking with images. h2 performs well for many small files but deteriorates for less or larger files. Sharding only helps h2 for larger files.

files. It is of note that the concatenated version is two to five times faster overall, even though (in a rare compression fluke) its file size is much higher than the sum of the individual file sizes. Additionally, we see that sharding deteriorates **h2**'s performance, while only marginally benefiting **h1** (right). Because the files are that small, **h2**'s multiplexing was at its best in the single host case and maximized the single connection's throughput, while in the sharded setting it has less data to multiplex per connection. Conversely, sharding empowers **h1** to open up more connections, but still suffers from HOL blocking on the small files.

Figure 1(b) shows relatively little differences and no clear consistent winners between the concatenated (left) and separate files (middle) over one host. This is somewhat expected for **h2**, as in both cases it sends the same amount of data over the same connection, but not for **h1**. We would expect the six parallel connections to have more impact, but it seems they can actually hinder on good network conditions. This is probably because of the limited bandwidth in our emulated cellular network, where the six connections contend with each other, while a single connection can consume the full bandwidth by itself. Unlike **h2**'s behaviour in (a), we see that here **h2** does not get significantly faster for the concatenated version. This indicates that the higher measurements in (a) (middle) are in large part due to the overhead of handling the many individual requests. Similarly, sharding (right) shows inconsistent behavior: sometimes it helps and sometimes it hurts **h2**; it shows impressive benefits for **h1c** but smaller gains for **h1s**. We posit that the additional overhead of setting up extra secured HTTPS connections (both for **h1s** and **h2s**) limits the effectiveness of the higher parallel throughput. Overall, we can state that there is no clear winner here, nor for the three different setups, nor for the three protocols.

Lastly, in Fig. 1(c) we see that **h2** struggles to keep up with **h1** for the larger files and performs significantly worse under bad network conditions (note the y-axis' log scale). Due to the much larger amount of data, **h1**'s larger amount of parallel connections do help here, while packet loss impacts the fewer **h2** connections more. This is immediately apparent when comparing the NoLoss and Good network conditions in Fig. 1(c) (middle): the **h1** measurements are very similar while the single **h2** connection is almost 80% slower in this case (note that the NoLoss and Good conditions are identical except for the amount of packet loss introduced). As expected, utilizing additional parallel connections (right) benefits both **h1** and **h2**, helping mitigate HOL blocking for **h1** and lessening the impact of loss when compared to a single **h2** host. The SpeedIndex measurements (not included here) show very similar trends for all of the experiments discussed in Fig. 1.

In conclusion, we can say that while **h2** indeed helps for many smaller files, it still loses to concatenated versions of those files, both over **h1** and **h2**. This indicates that the current **h2** implementations can incur heavy costs for handling individual resources, though this primarily poses a large problem for many (>42) files (see also Sect. 4.3). We can also conclude that **h2**'s single connection setup seems to suffer from bad network conditions, but not excessively more than **h1**, and the performance drop largely depends on the observed case. Similarly, we

have observed that using multiple parallel connections for **h2** can help mitigate this problem (especially for websites with large objects), but that it can also lead to slower load times (for many, smaller objects). These findings are consistent with the previous work of Goel et al. [10], who overall observed that *if* sharding helps for **h2**, sharding over more hosts helps more, but there are diminishing returns with each increase in the amount of hosts. Additionally, Mi et al. [21] decisively show that large files can increase the time to download smaller files by 99% over a single **h2** connection. They propose an extension to **h2** that allows migrating resource requests between parallel TCP connections (also in a multipath TCP setting). Interestingly, Manzoor et al. [16] have shown empirically that various browsers are already using multiple parallel connections for **h2** in the wild (although this was never observed during our tests). This might indicate that the browser vendors are aware of the beneficial nature of this practice. However, to the best of our knowledge, the browser vendors have yet to present their own results on this issue.

4.3 HOL Blocking in Practice with CSS and JavaScript

HOL Blocking with CSS and JavaScript with `loadEventEnd`. The discussion in Sect. 4.2 has clearly shown the impact of network conditions and the amount of parallel connections of **h2**'s performance. It has also shown that due to HOL blocking, **h2** seems to shine when loading a large amount of smaller files, but that it is not necessarily faster when the amount of files is lower. In order to investigate this property further and determine the point where HOL blocking is overcome, we observe two experiments in Fig. 2: 500 `<div>`-elements are styled using (left) simple CSS files (single CSS rule per `<div>`) and (right) complex JS files (multiple statements per `<div>`). We vary the degree of CSS and JS code concatenation, from one file (full concatenation) to 500 files (no concatenation). Figure 2 plots full results in (a) and shows more detail for one to 30 files in (b). We resorted to CSS and JS files in these experiments instead of images because they typically include additional processing from the browser, which can also impact page load time performance, as we will see. The data shown here is from tests using the Good network condition.

The big-picture trends in Fig. 2(a) look very similar to Fig. 1(a) (left and middle): **h2** again clearly outperforms **h1** as the number of files rises and shows a much better progression towards larger file quantities than the quasi linear growth of **h1**. Interesting is also the performance of Firefox: while its **h1** results (not shown in Fig. 2 for clarity) look almost identical to Chrome, its **h2** values are much lower, indicating that it has a more efficient implementation that scales better to numerous files.

Looking at the zoomed-in data in Fig. 2(b), we do see somewhat different patterns. For the simple CSS files the trends are relatively stable, with **h1c** outperforming **h2** and **h2** beating **h1s**. This changes at about 30–40 files, where **h2** finally takes the overall lead. For the more complex JS files (right), this tipping point comes much later around 100 files. The measurements for one to ten JS files are also much more irregular when compared to CSS. Because **h1**

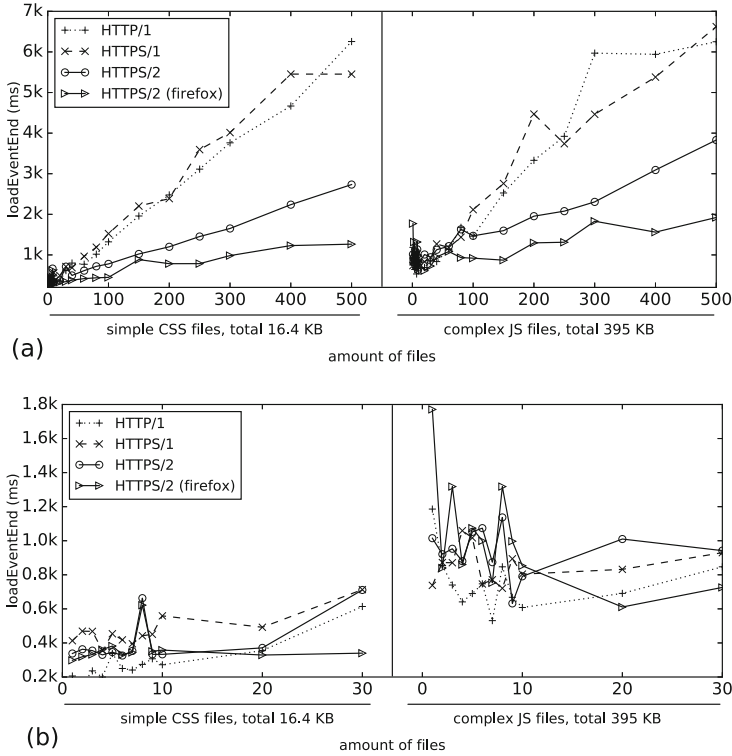


Fig. 2. Synthetic test cases for HOL blocking with CSS/JS files. **h2** performs well for many files but there is no clear winner for the more concatenated cases. Image taken from our previous work [18].

shows the same incongruous data as **h2**, we can assume this can be accounted to the way the browser handles the computation of the larger incoming files. The performance of a multithreaded or otherwise optimized handling of multiple files can depend on how many files are being handled at the same time. This would also explain the very high **h2** measurements for a single JS file in Firefox (consistent over multiple runs of the experiment). In additional tests, smaller JS files and larger CSS files also showed much more stable trends, indicating that especially large JS files incur a large computational overhead. Note as well that the timings for a smaller amount of JS files are sometimes higher than those for the larger amounts, indicating that concatenation might not always be optimal here (for none of the protocols). Poor network conditions (not shown here) show similar trends to Good networks, but the **h2** tipping points come later: 40–50 files for simple CSS, 150 for complex JS.

All in all, we can see that **h2** only overcomes **h1**'s HOL blocking problems at a relatively large amount of individual files (30+ in the best case). While most websites do include that many resources, our results also show that concatenating

files together (thus again reducing the total resource count) can overall be faster than sending individual files for all protocols (especially for CSS files and many images, see Fig. 1(a)). This again confirms our earlier thesis that browsers introduce a lot of overhead per individual resource/request, regardless of the actual size of the data (though Firefox seems to have a more efficient implementation than chrome, at least for h2) and that this issue needs to be resolved first before h2 can overtake h1 and its best practices.

HOL Blocking with CSS and JavaScript with SpeedIndex. For the tests in the previous Sect. 4.3, the `SpeedIndex` results were significantly different from the `loadEventEnd` measurements and merit separate discussion. Figure 3 shows the same experiment but depicts `SpeedIndex` for Google Chrome. We notice that the data for the simple CSS files (left) looks very similar to Fig. 2, but the results for the complex JS files (right) do not. Since the `SpeedIndex` metric gives an indication of how progressively a page renders (Sect. 3) and because we know from Fig. 2 that h1 takes much longer than h2 to load large amounts of small files, we can only conclude that under h2 the JS files take much longer to have an effect on the page rendering, to skew the `SpeedIndex` in this way.

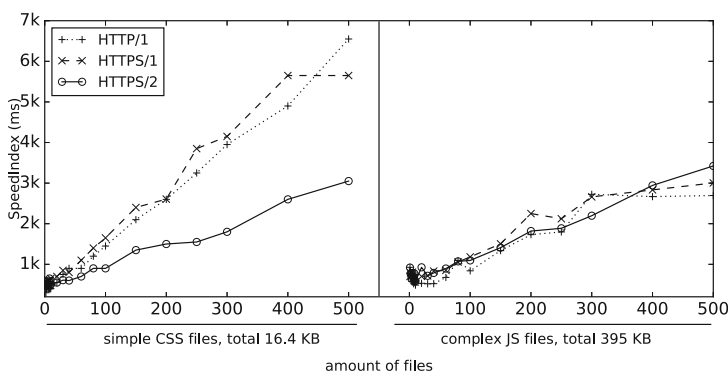


Fig. 3. Synthetic test cases for HOL blocking with CSS/JS files (`SpeedIndex` metric). h2 `SpeedIndex` for JavaScript indicates that it is much slower to start rendering than h1. Image taken from our previous work [18].

We manually checked this assumption using screenshots and found that for h1 the JS was indeed progressively executed as soon as a file was downloaded, but with h2 the JS code was applied in “chunks”: in larger groups of 50 to 300 files at a time and mostly towards the end of the page load. We first assumed this was because of erroneous multiplexing: if all the files are given the same priority and weight, their data will be interleaved, delaying the delivery of all files (see Sect. 5). Captures of h2 frame data in Google Chrome however showed that each file i was requested as dependent on file $i - 1$, and that file data was fully delivered in request order (consistent with the behaviour described

in Sect. 5). We can once more only conclude that the browser implementation somehow delays the processing of the files, either because of their JS complexity or because the handling of many concurrent **h2** streams is not optimized yet. This argument is supported by the **SpeedIndex** results for Firefox (not shown here, for clarity), as its **h2** values are much lower than those of **h1**, indicating that Firefox has a more efficient **h2** implementation than Chrome.

If the browsers’ handling of CSS code would be similar to that of JS code, we would expect to see similar results in Fig. 3 (left) and (right). However, if the CSS files would also be applied individually as soon as they were downloaded, the **h1** **SpeedIndex** values would be much lower than the observed measurements. We found that the browser delays execution of *all* CSS until they have all been downloaded and processed for both **h1** and **h2**, despite our experiments having been built specifically to prevent this. This is again unexpected browser behaviour (though probably not directly related to the **h2** implementation) and we plan to look deeper into this in future work, as discussed in [18].

5 Resource Prioritization

5.1 Background

As demonstrated in Sect. 4, **h2** solves the **h1** Head-Of-Line blocking problem by allowing multiple resources to be sent on the same connection at the same time. To make this possible, each resource is assigned to its own conceptual “stream” and these streams are then multiplexed over the single underlying TCP connection. The data from the individual files is split up in chunks and can thus be interleaved with chunks from other files. This is especially interesting for resources that are partly directly available but that need slow I/O operations to complete (e.g., an HTML template that fetches content from a database). Using multiplexing, chunks from other resources can be sent while the data of the “delayed” resource is being fetched, resulting in less idle time on the TCP connection. Alternatively, when concurrently sending a very large and a very small file, the data from the small file might be multiplexed with parts of the larger file so the receiver does not need to wait for the larger file to be fully downloaded to receive the smaller resource [21].

To this end, the **h2** specification [3] details the concept of a “dependency tree”. Nodes in this dependency tree represent individual **h2** streams, while the root of the dependency tree denotes the underlying TCP connection. New nodes are added to the tree as new resources are requested and nodes can be removed when their corresponding resources have been fully downloaded. A parent-child relationship between nodes indicates that the child’s transmission should be postponed until its ancestor has been downloaded completely (or until it is temporarily impossible to make progress on the parent resource). Conversely, a sibling relationship between nodes allows bandwidth to be distributed among the siblings proportionally to their “weight” (i.e., $\in [1,256]$), thus allowing multiple resources to be interleaved in a very fine-grained way. The **h2** buildup of the tree is decided by the browser at runtime and communicated to the server using

HEADERS or PRIORITY frames. This general setup allows for a lot of flexibility in how the dependency tree is effectively constructed and maintained by the browser during the page load.

Figure 4 shows example dependency trees from Google Chrome (a) and Mozilla Firefox (b) respectively. It is apparent that Chrome chooses a very sequential setup, where each node is the only child of its parent (rendering individual stream weights effectively useless). It does however maintain an internal “priority order” depending on the type and location of the resource (e.g., a CSS file in the <head> will be given a **Highest** priority level, while an image in the <body> will have a **Low** overall importance). If a new resource is discovered, it will be not be added at the end of the full tree, but rather after the last existing resource with the same priority level. Firefox utilizes similar priority bins internally (indicated by e.g., **leaders**, **followers**, **unblocked**) but chooses to build its priority tree in a radically different way from Chrome. Firefox adds “ghost” nodes for each of these priority levels (which do not directly represent an h2 resource or stream) to be able to group the h2 streams that belong to this category as siblings. This allows Firefox to use a more complex prioritization strategy for its h2 implementation.

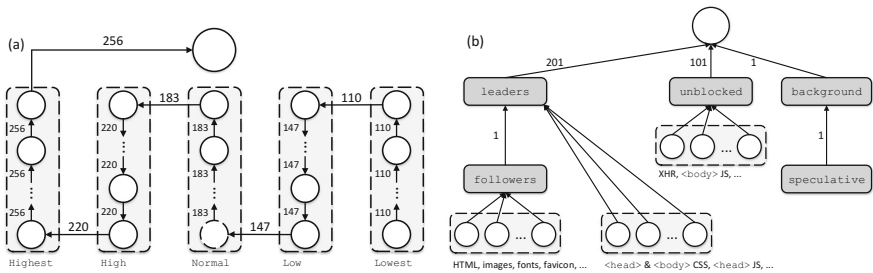


Fig. 4. HTTP/2 dependency tree layout of (a) Chrome and (b) Firefox. Numbers indicate node weights.

5.2 Evaluation of Prioritization Strategies

As Chrome’s and Firefox’s approaches for the h2 dependency trees are fundamentally different (Fig. 4), it is difficult to directly compare both options and see which one performs best. This is due to the fact that the both browsers are internally optimized for their specific strategy, implying that using a different strategy will skew the results. Instead, we implement two alternative, less complex prioritization strategies to see how much better (or worse) the browser’s more advanced approach works.

The first alternative algorithm, Round Robin (RR), is the default behaviour specified in the h2 specification [3]. All h2 streams are made siblings under the root node and each is given an equal weight. In effect, this causes all active

resources to be given an equal share of the bandwidth and leads to heavy multiplexing. The second algorithm, First-Come-First-Served (FCFS), approaches the way `h1` works. The dependency tree is purely linear and each new node is added as the bottom leaf node; FCFS is thus a much less advanced version of Chrome’s strategy as it will never inject new nodes between two existing nodes. FCFS entails that the current resource has to be sent fully before (any part of) the next resource can be sent, effectively disabling multiplexing.

These two alternative algorithms are implemented by modifying the `H2O` server source code. The server simply ignores the priority directives from the browser (which is allowed behaviour as per the `h2` specification [3]) and builds its own dependency tree using the rules described above.

Figure 5 shows the CDF results for the Firefox browser in Poor network conditions. We take a corpus of 40 websites (see Sect. 8 for details) and load them 20 times with each protocol/prioritization strategy; the median values are used in the CDF.

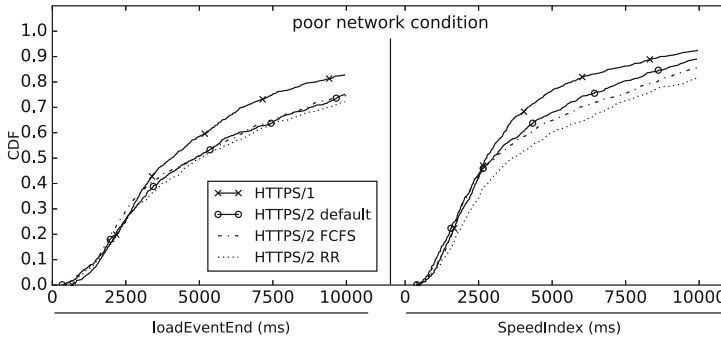


Fig. 5. HTTP/2 prioritization strategies in Firefox for a Poor network condition. Round Robin is clearly detrimental for `SpeedIndex` and `h1s` is consistently faster than `h2`.

We can deduce that the impact of the different `h2` prioritization strategies is moderate for the `loadEventEnd` metric, with `h2` measurements being very similar. However, the `SpeedIndex` metric clearly shows worse performance when using the Round Robin strategy. This is expected, as it will take longer for resources to be fully downloaded and because, as we have discussed in Sect. 4.3, browsers will often wait for a full resource (or group of resources) to be downloaded before re-rendering the page. This is a remarkable result because, as mentioned before, Round Robin is the default prioritization behaviour prescribed by the `h2` specification and in our tests it was seen in effect in both Microsoft’s Edge and Apple’s Safari browsers (which do not seem to employ a custom prioritization strategy at this time).

We performed similar tests for Google Chrome and on the various other network emulation settings detailed in Sect. 3. The results displayed in Fig. 5

are among the most distinct of all our evaluated data, meaning that other tests showed even less differences between the different strategies, especially for improved network conditions. This indicates that the adopted prioritization strategy is not a major influencer of h2 page load performance, except on poor networks and then only when used in the most straightforward way. This reflects previous work by Bergan [4], who found that Chrome’s implementation only clearly outperforms a completely random strategy in 31% of the observed pages.

Finally, when looking at the browsers’ prioritization strategies, we found that their implementations are often still lacking in their support of cutting-edge web technologies. For example, the new Service Worker concept⁷ allows developers to register a JS-based “client-side proxy” that can intercept and perform custom processing on all requests the browser emits. We found that all h2 requests passing through such a Service Worker lost all of their intelligent prioritization information, leading to Chrome defaulting to a FCFS-alike strategy, while Firefox exhibited pure RR behaviour. This is probably an implementation oversight and we expect this to be fixed in the future. As another example, developers can indicate to the browser that certain JS files are less important using the `async/defer` attributes. Chrome correctly assigns a Low priority to those resources, but in Firefox they are regarded as normal, high-priority JS files. We believe that these and similar browser implementation errors could be responsible, at least partly, for some contradictory results in other work (Sect. 2).

6 Server Push

6.1 Background

In HTTP/1.1, the browser can only receive resources that it has explicitly requested. Typically, the user agent first fetches the HTML page (e.g., `index.html`), which it then parses to discover other referenced resources. As such, it takes at least one RTT before the browser can start requesting critical CSS/JS files and a minimum of two RTTs before they are downloaded. Especially on slow networks, this can have a large performance impact. In response, the HTTP/2 specification [3] describes a novel mechanism called “Server Push”. This allows the server to decide to send along additional resources with previously requested resources, not having to wait for the browser to request them first, thus potentially saving a full RTT.

In theory, developers could push all necessary resources of a website along with the original HTML request and thus eliminate additional RTTs completely. In practice however, Server Push is limited by TCP’s congestion control mechanisms. For example, in its “slow start” phase, TCP sends only a small amount of data at the beginning of the connection and then exponentially ramps up its speed if no packet loss or delays are present. In practice, the TCP congestion

⁷ <https://developers.google.com/web/fundamentals/getting-started/primers/service-workers>.

window starts at about 14KB for modern Linux kernels (as used in our experiments) [17], severely limiting the amount of resource data we can push during the first RTT. Given this behaviour, h2 Server Push’s benefits should increase the longer the TCP connection stays open (i.e., the congestion window grows as the connection gets “warmer”), as more data can be pushed in a single RTT.

Server Push could be a good fit for the popular modern Single Page App (SPA) setup. In this paradigm, the loaded page routinely requests additional data from the server using a (REST) API, thus keeping the TCP connection active. The API’s response will then no longer just consist of the structured `xml/json` data, but can also contain the pushed subresources (e.g., images) mentioned in the data. Another interesting use case is to deploy Server Push from a network intermediary, such as a CDN proxy. In this setup, the browser typically connects to the proxy, which in turn connects to the origin server. The proxy can then “warm up” its connection to the browser by pushing static assets (mainly CSS/JS/font files) while it waits for the dynamic HTML and other data to arrive from the origin. This use case is discussed in-depth by Zarifis et al. [30], who show up to 27% web page load time improvements when using Server Push in this fashion.

However, the page load performance of Server Push can be very dependent on knowledge of the correct priority of the resources it wants to push and how they fit into the page loading process. The main reason for this is that large parts of common network stack implementations use buffered I/O, both on the OS-level and in the network itself [5]. Once data is queued in these intermediate buffers, it is often impossible to remove it or replace it with other data. If we then, for example, would immediately push three very large image files along with the initiating request and their data fills up all buffers, the browser’s request for a more critical CSS file will be delayed because we cannot re-prioritize the less important image data in the buffers. This makes for large practical difficulties in determining which resources to push and when [30]. This problem is enlarged by the fact that the h2 specification [3] does not include a mechanism for the browser to signal to the server which files it has already cached. Consequently, the server will potentially push files which the browser already has, wasting bandwidth and delaying other resources.

6.2 Experimental Evaluation

While a full in-depth evaluation of the discussed characteristics of h2 Server Push is out of scope for this work, we can nevertheless demonstrate several of the discussed aspects using a very simple example. We use the existing Push demo by Bradley Fazon⁸, based on the www.eff.org frontpage. This page has a single “critical CSS” file (which is responsible for the main look-and-feel of the site and thus a good Push target) and a good mix of additional CSS, JS and image files without being too complex. We make sure the initial HTML code is smaller than

⁸ <https://github.com/bradleyfazon/h2push-demo>.

14KB (by removing some metadata and enabling gzip compression), reducing the on-network HTML size from 42KB to 9KB.

Figure 6 shows the results from tests using the Apache webserver because NGINX does not yet support h2 Server Push. The SpeedIndex results are displayed because these should be most affected. We observe five different experiments: (1) push the single “critical CSS” file, (2) push all the CSS/JS files (10 files), (3) push all (images + CSS/JS + one font), (4) push all images (18 files) and (5) the reference measurement (original, no push). We see that pushing the one “critical CSS” file does indeed improve the SpeedIndex measurements, but not excessively so. It is unexpected however that pushing all CSS/JS performs a little better than just the “critical CSS” in the Good network condition. We found that in practice the initial data window is often a bit larger than 14KB, so it can accommodate more than just the single CSS file. However, this is not always the case and other runs of the same experiment show less optimal results (which can also be seen in the Poor network condition). Given a larger data window, the “Push all” test case should perform similarly to pushing the CSS/JS resources, but it is consistently a bit slower. It turned out that we pushed the single font file at the very end, after all the images. The font data should have been given a higher h2 priority than the images, but due to an Apache bug⁹ this was not the case and the font data had to wait, delaying the final render. This also explains why pushing just the images performs worse than the reference: the much more important CSS and JS is delayed behind the image data.

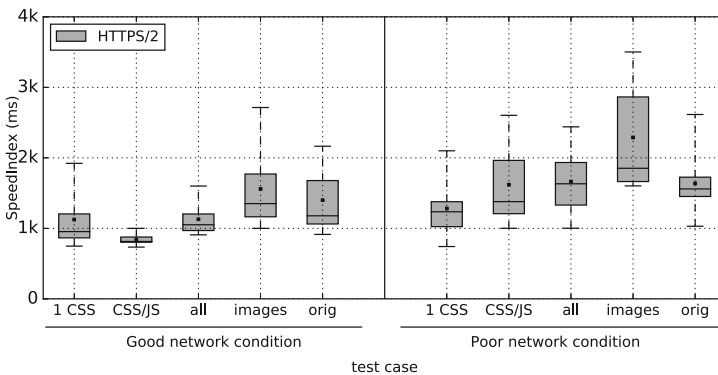


Fig. 6. Realistic test case for HTTP/2 Push. Pushing the wrong assets or in the wrong order can deteriorate performance.

The discussed aspects and challenges make Server Push difficult to fine-tune to achieve optimal performance. Due to this and the fact that many popular h2 server implementations do not yet support Server Push, Zimmerman et al. [31] found that out of their observed 5.38 million HTTP/2 enabled domains, only 595 actively used Server Push.

⁹ https://icing.github.io/mod_h2/nimble.html.

7 HPACK Header Compression

7.1 Background

HTTP uses the concept of *Headers* to convey various types of metadata about its requests and responses between the user agent and the server. These headers are typically prepended to the actual message body. Some popular header names are `Content-Type`, `Keep-Alive`, `Cache-Control` and `Cookie`. This last header is useful to bind multiple requests and responses to the same conceptual “user session”, allowing applications to provide stateful interactions. Cookies typically contain a numeric user ID or session token but can also include more complex (serialized) data, which can make them relatively large in practice [29].

The headers are often repeated with each individual message, which can be wasteful with respect to bandwidth usage, especially in the case of large metadata like `Cookie`. HTTP/2 attempts to solve this deficiency by introducing HPACK [24], a compression algorithm specifically tuned to the HTTP header format. HPACK combines a pre-defined dictionary of known prolific header names and values with a dynamic shared dictionary per connection that is built up at runtime (at both the server and browser side), based on the header data that is actually being sent during the session. As such, HPACK will perform better with large amounts of similar files or cases with large dynamic metadata, as it learns the repeating data on-the-fly. For example, the first time a header of the form `Cookie: value` is sent on the connection, it is stored in the dynamic dictionary. The next time this specific header would be sent, it can be wholly replaced by a reference to the dictionary entry, which is identical on both client and server.

7.2 Experimental Evaluation

To demonstrate the behaviour of HPACK, we use data gathered during our experiments from Sect. 4, which include only typical HTTP headers and no cookies are set. Table 2 details three cases: (a) 10 large images, (b) 42 medium images and (c) 400 complex JS files. The `BytesOut` measurements consist of all data that was sent by the browser and thus include primarily HTTP request headers and TLS connection setup data. The actual header-induced overhead is even larger if we also consider HTTP response headers.

For the cases with one host server, we can clearly see that HPACK significantly reduces the overall header size when compared to `h1`, with a factor of more than five for case (c). It is also apparent that the header overhead is typically relatively low but can grow to 27% for many individual files on `h1s` (as each of those files requires a separate request and response message). Looking at the sharded setup with four host servers, we see that while both protocols produce more overhead from the extra connections, `h2s` relatively suffers more than `h1s`, especially for (a) and (b) (note that the `h1s` overhead is mainly due to the TLS overhead from opening 24 connections compared to the six for the single host case and four for sharded `h2s`). This is expected, as `h2s` now has

less data to learn from on each individual connection and optimize its dynamic compression scheme. This is another argument of h2 to favor using only a single underlying TCP connection.

It is of note that these observed header compression results are arguably too low to have a significant impact on the performance of any individual page load of a realistic website. However, when viewed on a larger scale (e.g., cumulatively across all the servers in a data center or CDN) these savings can add up and make a significant difference in the overall bandwidth usage of popular websites. Related work from Cloudflare [1] indicates that on average HPACK reduces HTTP header size by 30% and overall HTTP/2 egress traffic by 1.4%, with outliers of up to 15% for individual websites.

Table 2. Total bytes sent by Google Chrome (~HTTP headers) and ratio to total page size. For many small files, the HTTP header overhead is significant. Sharding over multiple hosts decreases the effectiveness of HPACK header compression.

File count	Protocol	Total page size	1 host		4 hosts	
			BytesOut	% of total page size	BytesOut	% of total page size
(a) 10 large files	h2s	2177600	504	0.02%	1227	0.05%
	h1s	2177600	2419	0.1%	2993	0.1%
(b) 42 medium files	h2s	1075000	649	0.06%	1362	0.1%
	h1s	1075000	2786	0.2%	3346	0.3%
(c) 400 small files	h2s	610000	29580	4%	38680	6%
	h1s	610000	165300	27%	177600	29%

8 HTTP/2 Performance for Realistic Web Pages

8.1 Experimental Setup

While the synthetic test cases from the previous sections (excluding Sect. 5) are useful to assess the individual h2 performance techniques in isolation, they are not always representative for real websites. We will now look at some more realistic test cases. We will first present results for a corpus of nine manually selected website landing pages (corpus A), which all contain either many smaller images (e.g., media/news sites) or fewer but larger images (e.g., product landing pages with large images taking up most of the “above the fold” space). The composition of this corpus is motivated by the goal of enabling easy and meaningful comparison with our synthetic experiments in Sect. 4.2. As we will see however, while the resulting findings showed clear trends, it was difficult to pinpoint their

underlying causes. In response, we executed additional tests on a second, larger corpus of 40 landing pages (corpus B) taken from the Alexa Top 50 and Moz Top 500 rankings¹⁰. These pages were selected primarily on their total filesize, with 10 pages being low-weight (<500 KB), 10 pages medium-weight (≥ 500 KB, ≤ 1 MB) and 20 pages heavy-weight (>1 MB). All pages were cloned using the `wget` tool¹¹ so that they could be served locally in the Speeder experimental setup (Sect. 3).

The experimental setup is meant to simulate what would happen if a developer would switch their `h1` site to `h2` by naively moving all their own assets over to a single server (disabling sharding) but still downloading some external assets from third party servers (e.g., Google analytics, some JS libraries). This approach is similar to the one adopted in [27]. We expect to see good `h2` performance compared to `h1`, as the latter has only six parallel connections to work with and `h2` can optimally use its single TCP connection.

The results for corpus A are from server NGINX v1.10, browsers Google Chrome v54 and Mozilla Firefox v49 and test runner webpagetest v2.19. Each page was loaded at least 10 times. The results for corpus B were obtained later during our research through the standard H2O server v2.1, Google Chrome v58, Mozilla Firefox v54 and webpagetest v3.0. Each page was loaded at least 20 times. We will display the median values. For more details on both test corpora and the Speeder setup, we refer to our website (see Sect. 3).

8.2 Experimental Results

Figure 7 shows the median `loadEventEnd` and `SpeedIndex` measurements for corpus A over Good and Poor networks. Globally, we can state that `loadEventEnd` and `SpeedIndex` are often similar for the three protocols on the Good network, indicating that the page load times of the tested pages are mostly network dependent, with the rendering having to wait for assets to come in. This explains why Poor network conditions can have a very large impact on page load time performance (see Fig. 7(right)). In various cases, `h2`'s `SpeedIndex` is far above that of `h1` even if their `loadEventEnd` values are similar, indicating that `h2` is slower to start rendering, consistent with our observations in Sect. 4.3. `h1c` is faster than `h2` in almost all of the cases and `h2` is almost never much faster than `h1s`. Note that this is somewhat against our hypothesis, as `h1s` has to make due without the benefits of sharding. A more in-depth discussion of some of the outliers in Fig. 7 can be found in [18].

Looking more closely at the results for Poor networks in Fig. 7, we see that `h2` is sometimes much slower than `h1` but sometimes is also relatively similar. Given the limited size of corpus A, it was difficult to pinpoint the underlying reasons for this inconsistent behaviour. Suspecting that the total page size and amount of objects on the page had a large influence (both from the corpus A results and our synthetic tests in Sect. 4), we ran additional tests on the larger corpus B. The

¹⁰ <http://www.alexa.com/topsites>, <https://moz.com/top500>.

¹¹ <https://www.gnu.org/software/wget/>.

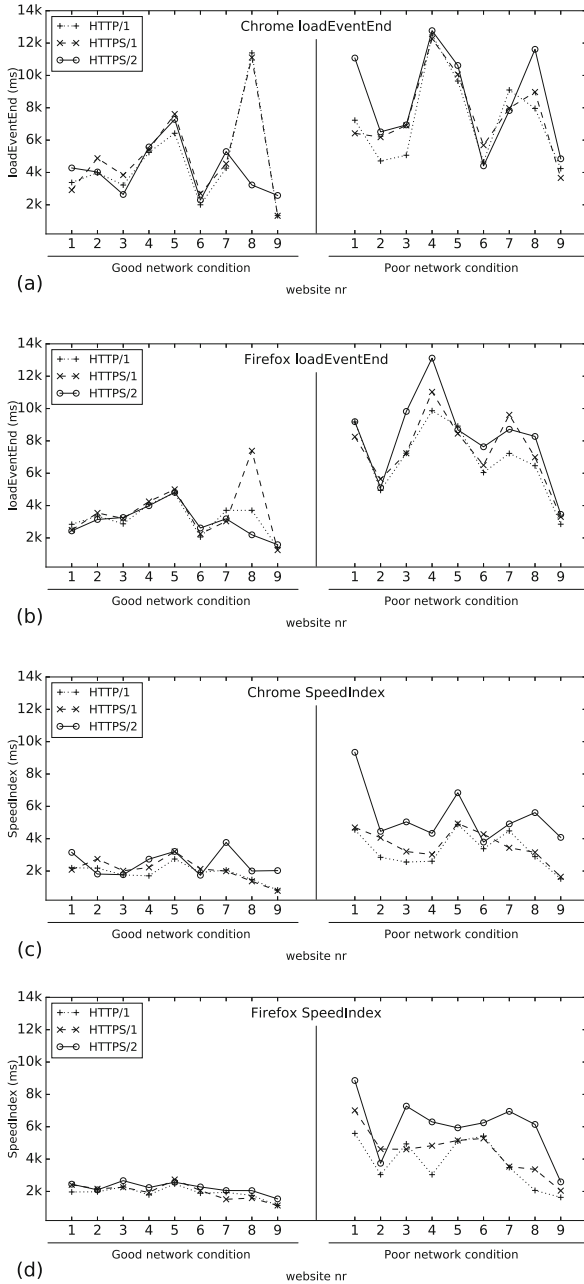


Fig. 7. Nine realistic websites from corpus A on the dynamic Good and dynamic Poor network models. There is very similar performance under Good network conditions, but h2 clearly suffers from Poor conditions. Image taken from our previous work [18].

results in Fig. 8 show that our thesis was indeed correct: the low-weight pages (left) have similar `SpeedIndex` performance for `h1s` and `h2s` even on the Poor network, while for the heavier pages (right) `h2` clearly suffers. The results for the `loadEventEnd` metric showed similar though less pronounced trends for the Poor network and only small differences in the three protocols’ measurements on the Good network.

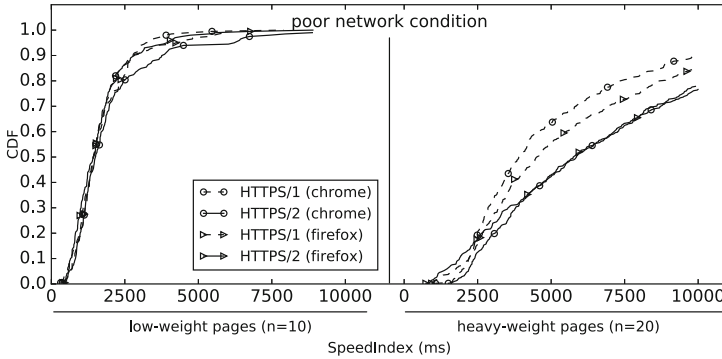


Fig. 8. Differences in `SpeedIndex` for low-weight and heavy-weight pages from corpus B in Poor network conditions. Heavy-weight pages clearly cause `h2` to suffer more and are faster to load under `h1s`.

Finally, it is difficult to directly compare our results for realistic pages to related work, since few authors present results from a large corpus of locally cloned web pages over various network conditions with modern `h2` implementations or for the `SpeedIndex` metric. The closest related work loads pages directly on the internet via various networks and shows more positive `loadEventEnd` results for `h2` than our tests do, for example that 80% of pages on faster networks clearly benefit from `h2` [26]. This percentage is lower on slower networks but there `h2` typically also has a higher benefit. We are unable to confirm their findings with our measurements. The most recent related work [31] loads pages over a high speed link and concludes that 51% of the tested pages are $\geq 5\%$ faster over `h2` when compared to `h1s`, which is also contradictory to our realistic test case results.

9 Discussion

Conceptually, the ideal HTTP/2 setup will use a single TCP connection to multiplex a large amount of small and individually cacheable site resources. This mitigates the HTTP/1.1 application-layer HOL blocking issue and helps to reduce the `h1` overhead of many parallel connections, while also maximizing the efficiency of the underlying TCP protocol. Together with advanced resource prioritization strategies, Server Push and HPACK header compression, this can lead to (much) faster load times than are possible today over `h1`, with less overhead.

Unfortunately, as our experiments have shown, this ideal setup is not yet viable. While **h2** is indeed faster than **h1** when loading many small files (Figs. 1 and 2), it is still often slower than loading concatenated versions of those files over **h2** (Sect. 4.2). Looking at the **SpeedIndex** metric results (Figs. 3, 7 and 8) also shows that **h2** is frequently later to start rendering the page than **h1**. HTTP/2 also struggles when downloading large files (Figs. 1 and 8) and its performance can quickly deteriorate when used in bad network conditions. In our observations, **h2** is in most cases currently either a little slower than or on a par with **h1** and shows both the most improvement and worst deterioration in extreme circumstances.

The good news is that almost all of the encountered problems limiting **h2**'s performance seem to be due to inefficient implementations in the used server and browser software. Firstly, while loading many smaller files incurs its own considerable browser overhead, the comparison of Chrome and Firefox in Fig. 2 tells us that this overhead can be reduced, as Firefox seems to have especially optimized its pipeline for large amounts of files. Secondly, the fact that **h2** is later to start rendering than **h1** is also due to ineffective processing of the **h2** data, since we have confirmed that resources are received well in time to enable faster first paints (Sect. 4.3). Thirdly, several cases in which **h2** underperformed could be attributed to the server or browser not correctly (re-)prioritizing individual assets (Sects. 5 and 6). As these implementations mature, we can expect many of these issues to be resolved.

However, **h2** still retains some core limitations, mostly due to its single underlying TCP connection, which seems to simultaneously be its greatest strength and weakness. TCP's congestion control algorithms can lead **h2** to suffer significantly from packet loss on Poor networks (most obvious when downloading multiple large files (Figs. 1 and 8)) and can heavily impact the effectiveness of **h2** on newly established connections (Sect. 6). We have to nuance these statements however, as in practice **h2** actually performs quite admirably and usually does not suffer more from bad networks than **h1**, despite using fewer connections. Additionally, we have found that **h2** can also benefit from using multiple connections in bad networks, especially in the cases where its performance problems are greatest.

The other discussed **h2** performance aspects do not seem to have as large an impact as the use of the single TCP connection. While prioritization is certainly important, the exact strategy that is used seems to have relatively little impact in most cases. Chrome and Firefox use wildly different algorithms to build their dependency trees (Sect. 5). Similarly, HPACK has only a limited impact on the total used bandwidth for most normal cases and will probably not directly affect individual page load times (Sect. 7). Finally, **h2** Server Push sounds like a powerful optimization but takes a lot of work and special network setup (e.g., CDN intermediaries) to save more than a single RTT on a page load (Sect. 6). Further work is needed to determine how to optimize both **h2** resource prioritization and Server Push.

Recognizing that the core h2 performance problems stem primarily from the use of TCP, the new QUIC protocol [6] implements its own application-layer reliability and congestion control logic on top of UDP. QUIC removes the transport-layer HOL blocking by allowing out-of-order delivery of packets, differently handles re-transmits in the case of loss, reduces the amount of round-trips needed to establish a new connection and allows larger initial data transmissions. Running h2 on top of QUIC could greatly benefit h2's multiplexing setup.

As such, we can conclude that the HTTP/2 protocol specification is a solid foundation for the next steps in bringing better page load performance to the web and reducing overall overhead. It will however take some time for implementations to mature and the QUIC protocol to be finalized before we will see its largest benefits in practice.

10 Conclusion

In this work we have discussed and evaluated four salient performance-related aspects of the new HTTP/2 protocol: using a single underlying TCP connection (Sect. 4), prioritization of multiple resources over this single connection (Sect. 5), the new Server Push construct (Sect. 6) and HPACK header compression (Sect. 7). Our evaluation was comprehensive and varied, looking both at synthetic and realistic test cases, over a variety of software, performance metrics and emulated network conditions.

Our results have shown that the switch to the single multiplexed TCP connection has by-and-large the biggest performance impact when comparing h2 to h1's multiple parallel connections. While in most cases h2 performs similarly to or slightly better than h1 (while inducing much less overhead), poor network conditions coupled with large files can cause h2's performance to deteriorate. The emerging QUIC protocol might help h2 overcome these problems by switching to UDP, while in the mean time using multiple concurrent h2 connections can also help.

Other discovered performance problems, such as h2 delaying the time to start rendering web page content, were likely to stem primarily from incomplete or erroneous h2 implementations and are expected to be solvable in time. Similarly, prioritization and Server Push both have potential but require future work to determine their best practices.

Acknowledgements. This work is part of the imec ICON PRO-FLOW project. The project partners are among others Nokia Bell Labs, Androme, Barco and VRT. Robin Marx is a SB PhD fellow at FWO, Research Foundation - Flanders, project number 1S02717N. Thanks to messrs Goel, Michiels, Robyns, Menten, Bonn e and our anonymous reviewers for their help.

References

1. Alpichi, K.: HTTP Pipelining. <https://blog.cloudflare.com/hpack-the-silent-killer-feature-of-http-2/> (2017). Accessed 08 Aug 2017
2. Beheshti, H.: HTTP/2: What No One's Telling You (2016). <http://www.slideshare.net/Fastly/http2-what-no-one-is-telling-you>. Accessed 01 Mar 2017
3. Belshe, M., Peon, R., Thomson, M.: HyperText Transfer Protocol Version 2 (2015). <https://tools.ietf.org/html/rfc7540>. Accessed 01 Mar 2017
4. Bergan, T.: Benchmarking HTTP/2 Priorities, October 2016. https://docs.google.com/document/d/1oLhNgIskaWD4_DtaoCxdSRN5erEXrH-KnLrMwEpOtFY/
5. Bergan, T., Pelchat, S., Buettner, M.: Rules of Thumb for HTTP/2 Push (2016). <https://docs.google.com/document/d/1K0NykTXBbbTlv60t5MyJvXjqKGsCVNYHyLEXIxYmV0>
6. Carlucci, G., De Cicco, L., Mascolo, S.: HTTP over UDP: an Experimental Investigation of QUIC. In: Proceedings of the ACM Symposium on Applied Computing, pp. 609–614. ACM (2015)
7. Corbel, R., Stephan, E., Omnes, N.: HTTP/1.1 pipelining vs HTTP2 in-the-clear: performance comparison. In: 2016 13th International Conference on New Technologies for Distributed Systems (NOTERE), pp. 1–6, July 2016
8. Erman, J., Gopalakrishnan, V., Jana, R., Ramakrishnan, K.K.: Towards a SPDY'ier Mobile Web? In: Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, CoNEXT 2013, pp. 303–314. ACM, New York (2013). <https://doi.org/10.1145/2535372.2535399>
9. Everts, T., Kadlec, T.: WPO Stats (2017). <https://wpostats.com/>. Accessed 03 Aug 2017
10. Goel, U., Steiner, M., Wittie, M.P., Flack, M., Ludin, S.: HTTP/2 performance in cellular networks: poster. In: Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking, MobiCom 2016, pp. 433–434. ACM, New York (2016). <https://doi.org/10.1145/2973750.2985264>
11. Gooding, M., Garza, J.: Real World Experiences with HTTP/2 (2016). <https://www.slideshare.net/JavierGarza18/real-world-experiences-with-http2-michael-gooding-javier-garza-from-akamai>. Accessed 01 Mar 2017
12. Grigorik, I.: High Performance Browser Networking. O'Reilly Media Inc, Sebastopol (2013)
13. Kohavi, R., Deng, A., Longbotham, R., Xu, Y.: Seven rules of thumb for web site experimenters. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1857–1866. ACM (2014)
14. Krasnov, V.: HPACK: The Silent Killer (feature) of HTTP/2 (2017). https://developer.mozilla.org/en-US/docs/Web/HTTP/Connection_management_in_HTTP_1.x#HTTP_pipelining. Accessed 08 Aug 2017
15. Liu, Y., Ma, Y., Liu, X., Huang, G.: Can HTTP/2 really help web performance on smartphones? In: 2016 IEEE International Conference on Services Computing (SCC), pp. 219–226. IEEE (2016)
16. Manzoor, J., Drago, I., Sadre, R.: The curious case of parallel connections in HTTP/2. In: International Conference on Network and Service Management (CNSM), pp. 174–180. IEEE (2016)
17. Marx, R.: HTTP/2 Push: The Details (2016). <http://calendar.perfplanet.com/2016/http2-push-the-details/>. Accessed 01 Mar 2017

18. Marx, R., Quax, P., Faes, A., Lamotte, W.: Concatenation, embedding and sharding: do HTTP/1 performance best practices make sense in HTTP/2? In: Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST 2017), pp. 160–173. INSTICC, ScitePress (2017)
19. Meenan, P.: Speed Index (2012). <https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index>. Accessed 01 Mar 2017
20. Meenan, P.: Webpagetest (2016). <https://webpagetest.org>. Accessed 01 Mar 2017
21. Mi, X., Qian, F., Wang, X.: SMig: stream migration extension for HTTP/2. In: Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT 2016), pp. 121–128 (2016)
22. Netravali, R., Goyal, A., Mickens, J., Balakrishnan, H.: Polaris: faster page loads using fine-grained dependency tracking. In: 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2016) (2016)
23. de Oliveira, I.N., Endo, P.T., Melo, W., Sadok, D., Kelner, J.: Should i wait or should i push? A performance analysis of push feature in HTTP/2 connections. In: Proceedings of the Workshop on Fostering Latin-American Research in Data Communication Networks. ACM (2016)
24. Peon, R., Ruellan, H.: HPACK: Header Compression for HTTP/2 (2015). <https://www.rfc-editor.org/rfc/rfc7541.txt>. Accessed 07 Aug 2017
25. de Saxcé, H., Oprescu, I., Chen, Y.: Is HTTP/2 really faster than HTTP/1.1? In: IEEE Conference on Computer Communications Workshops (INFOCOM), pp. 293–299. IEEE (2015)
26. Varvello, M., Schomp, K., Naylor, D., Blackburn, J., Finamore, A., Papagiannaki, K.: Is the web HTTP/2 yet? In: Karagiannis, T., Dimitropoulos, X. (eds.) PAM 2016. LNCS, vol. 9631, pp. 218–232. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30505-9_17
27. Wang, X.S., Balasubramanian, A., Krishnamurthy, A., Wetherall, D.: How speedy is SPDY? In: NSDI, pp. 387–399 (2014)
28. Wang, Z.: Navigation Timing API (2012). <https://www.w3.org/TR/navigation-timing>. Accessed 01 Mar 2017
29. Yue, C., Xie, M., Wang, H.: An automatic HTTP cookie management system. *Comput. Netw.* **54**(13), 2182–2198 (2010)
30. Zarifis, K., Holland, M., Jain, M., Katz-Bassett, E., Govindan, R.: Making effective use of HTTP/2 server push in content delivery networks. Technical report, University of Southern California, Networked Systems Laboratory, January 2017
31. Zimmermann, T., R uth, J., Wolters, B., Hohlfeld, O.: How HTTP/2 pushes the web: an empirical study of HTTP/2 server push. In: 2017 IFIP Networking Conference and Workshops (2017)



CUBE System: A REST and RESTful Based Platform for Liquid Software Approaches

Clay Palmeira da Silva^(✉), Nizar Messai^(✉), Yacine Sam^(✉),
and Thomas Devogele^(✉)

Université de Tours, 30 Avenue du Monge, Tours, France
clay.palmeiradasilva@etu.univ-tours.fr,
{nizar.messai,yacine.sam,thomas.devogele}@univ-tours.fr

Abstract. In the last years, a multi-host environment, that means, having at our disposal multiple connected devices, had become common in our daily lives. We are inundated with many services and applications with a promise to enhance our daily. However, the opposite occurs. We spend too much time and effort to use the same services in all multi-hosts we own. For dealing with this gap, we propose a model based on some principles of Service-Oriented Architecture (SOA) to achieve Liquid Software concepts. This approach requires a service design with an innovate and a unified interface which can drift between different connected devices and bring us with its behavior and complexities. In this context, based on REST and RESTful principles and constraints we propose an innovate centric-basic architecture to deal with multi-hosts. The CUBE, once finished, will allow that both services and multi-hosts discovery for migration across its system to enhance the usability of all devices owned by the same user.

Keywords: REST · RESTful · Multi-hosts · Services
Liquid software · CUBE

1 Introduction

In the last years, a multi-host environment, that means, having at our disposal multiple connected devices, had become common in our daily lives. Each device differs from others regarding a processing and storage capacities. Moreover, we spend too much time and effort to use the same services in all multi-hosts what we owned.

In this scenario, we find the same service or software provided from different kinds of platforms. Thus, the user must repeat her/his efforts each time when trying to use those services or software on all different devices.

One attempt to make user daily enhanced with all these different services and platforms, a Service-Oriented Architecture started to be adopted to unify

services. For this, developers have been widely adopted to use the concept of RESTful, and a large number of APIs or frameworks have been created and defined as so, although, several studies indicate that many APIs are not fully RESTful [9].

RESTful principles require the use of hypermedia. Thus, API or Frameworks which apply XML or JSON on their codes do not make use of hypermedia handling, since these languages are not able to deal with hypermedia [16]. Furthermore, to be defined as a Level 3 RESTful API means to have the ability to change the current state of a resource through hypermedia. Thus, according to [16], APIs which do not deal with hypermedia are not fully mature and cannot be defined as RESTful.

Thus, this leads us to another insight: Is the HTTP protocol used entirely and correctly? [17], Is the REST architecture respected? In other words, are the required design constraints defined by [16], followed to achieve RESTful principles?

Initially, these unanswered questions still require further investigation in our research. Meanwhile, other related issues drive us in different directions, such as: (i) Maturity Software [15]; (ii) Instantiation Resources [16, 17]; (iii) HTTP Link Header [2, 14]; (iv) API Conversation [9] and (v) Liquid Software [1, 12].

These directions lead us to assess how Web Services are built. While they exchange representations of their resources with consumers, they never provide direct access to the actual state of the underlying resources [15]. Therefore, the Web does not support pointers [15], and URIs are used to construct all representations with their resources on the Web, relating, connecting and associating them.

All these different approaches and techniques are driven towards a common interest, of attempting to find a way to enhance communications through API, Web Services, and Frameworks, using or not the concepts of Liquid Software, detailed later in this paper. However, an approach or methodology which can combine these techniques to achieve results is still unavailable.

It is in this scenario that we present our CUBE model based on a user-centric architecture approach, to support multiple connected devices owned by the same user. Once the information is acquired from a Server, it is possible to share data with all the connected devices in the same user-network through a trigger. Thus, the approach is not only a technology integrator but also a way of saving resources.

This paper is an extension of [20]. It describes the concepts of our CUBE system for the design of Web applications and how to extend current technologies to support its novel requirements. The paper is structured as follows: Sect. 2 provides background regarding the previously mentioned main constraints. Section 3 provides extended discussions regarding our findings while Sect. 4 describes the Use Case system diagram. Section 5 deals with the description of the Implementation Model. Section 6 gives concluding remarks highlighting future research directions.

2 Liquid Software Approach Motivations

This section describes how different techniques and perceptions led us to our proposal, and how we extracted the best state-of-art practices and built them into our CUBE model.

2.1 RESTful Principals and Constraints

We can assume that, while service-oriented computing becomes adopted every day, SOAP-based APIs is more and more giving place for REST-based approaches. According to [2] in 2014, 2125 SOAP-based APIs were developed, against 6833 REST-based ones.

To be defined as RESTful, the platform should follow some principles, as described in [16], which, in general, lead to high system performance, reduced consumption, high interoperability and security in all devices, as well as encapsulated legacy systems.

In addition, RESTful services should also respect some constraints, such as: (i) Stateless interaction [16], (ii) Resource linking, (iii) Identification or addressability of resources [2], (iv) Uniform interface, (v) Hypermedia as a mechanism for decentralized resources and (vi) Self-describing messages [14].

Thus, when all these principles and constraints are put together as best practices to create a REST-based application (API or Framework), another problem emerges: Which language can be used to support a level 3 RESTful characteristic? What should be the choice to support hypermedia controls? These questions lead to the concept of maturity software addressed in the next subsection.

2.2 Maturity Software

A level 3 RESTful characteristic relies on the ability of a service to change the set of links that are given to a client based on the current state of a resource [11, 16]. In this context hypermedia support controls, XML and JSON [22] are not able to achieve a real Level 3. On the other hand Atom, XHTML or JSON-L can do so [16].

However, what exactly does this mean? An API to be defined as RESTful should be fully mature, thus, making use of hypermedia to model the relationship between resources. As put simply by [15] that hypermedia drives systems to transform their application state.

Still, according to [15], the hypermedia system is described by participants as a transferring resource representation that contains links according to an application protocol.

The maturity level of service also affects the quality attributes of the architecture in which the service is embedded [16]. That is why a standardized and uniform interface was applied to each resource to remove unnecessary variations and to enabling all services to interact with all resources within the architecture, thus promoting interoperability and serendipitous reuse [16, 23].

2.3 Instantiation Resources

In a scenario of multiple devices, the instantiation resource feature is significantly essential. As a necessity of most RESTful Web Services, which enables clients to create new resource identifiers and set the corresponding state to an initial value [16].

Consequently, a resource can either be set by the service or by the client. Thus, it is possible to assume that the URI created by the service is unique, while it is possible that multiple clients generate the same identifier [16]. This feature can be used to provide a new set of services delivered by different servers to the same client. However, to do this, the approach proposal described by [12] should be modified for an inside perspective, the connected devices share the services.

In this aspect, we allow for the use of the POST, GET, SET and DELETE to be managed by the API to achieve the result expected by the client. Thus, it seems necessary to establish a better appliance of semantics concepts, not only in the Instantiation Resources but also in the Link Header, which changes significantly according to client behavior.

However, when accessing a social media platform such as LinkedIn through Facebook or Google, authorization from the client is required, and these platforms often accuse a security error when attempting integration. It is noteworthy that the certificate to access the different resources is not provided by the Servers/Services, but by the client through an encrypted password.

2.4 Link Header

A Link header is a powerful HTML resource, although not usually correctly applied. It gains significant value in REST-based approaches. Some exciting research has been conducted in this regard, as seen in [10], which proposes to discover REST resources as the user navigates. Their work seems promising but depends too much on perfect assumptions to obtain results.

Another similar approach is composed of RESTful-linked services on the Web, as proposed by [2]. Despite their effort and proposal, there is a lack of concern with the use of Semantics to achieve better results.

In fact, when the link header is used correctly, it is possible to include one or more hyperlinks. Thus, multiple targets are shown to be selected by the client. This solution gives rise to other issues, e.g., which URI should the client follow and does its selection imply the expected result by the client? If so, how should the consumer's choice of the URI be conducted to obtain the expected outcome?

These issues still require attention to enhance the Link Header. It is essential not only to flood the consumer with all URIs but also to give a well-defined set of useful URIs.

2.5 API Conversation

We can define an API Conversation as a set of communication activities between two or more participants [9]. Thus, the focus is on communication between a consumer and a RESTful Web API. From the client’s perspective, this means an interaction with a particular API to fulfill a goal. Thus, the resources are the building blocks of each RESTful Web API [9].

When a consumer requires multiple request/response interactions, she/he wishes to exchange published resources with one or more consumers. Therefore, it is possible to have various RESTful APIs emerging from the navigation within a Web through hypermedia relationships sponsored by consumers [2, 9, 12, 16]. Figure 1 illustrates how multiple requests can occur at the same time.

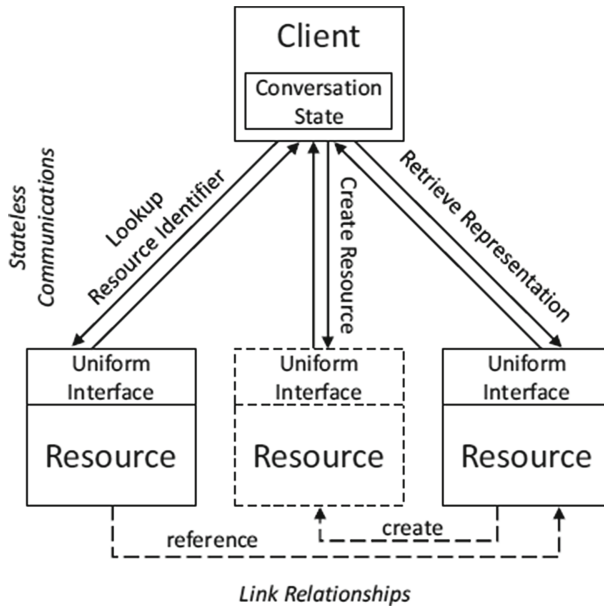


Fig. 1. How a client can use data from different services. Figure from [9].

However, in this scenario, still according to [9], four examples of RESTful conversation types were collected: (i) Redirect, (ii) Accessing Resources Collections (iii) Try-Confirm-Cancel, and (iv) Long Running Requests. Each example has a different way to deal with the RESTful conversation.

In this context, was identified another issue: A model API which combines multiple basic conversation types. That means, how we can use semantics as mediation support (repository) for dealing with multi REST API conversations. Furthermore, we can use semantics to (i) Build, (ii) Read, (iii) Evaluate, (iv) Compare and (v) Enhance multiple REST APIs conversations.

2.6 Liquid Software

Despite being a technology mentioned at the end of the 90s [8], over two decades ago, we have seen no significant enhancements in this method. However, one thing has changed, our behavior, since most of the time, we use at least two Internet-connected devices at the same time.

When we expand this perspective for the available possibilities, such as laptops, smartphones, tablets, phablets, game consoles, smart TVs, car display, watches, augmented reality glasses, digital cameras and photo frames, home appliances and so on [1], we obtain a perspective of insufficient resources to deal with this amount of devices at the same time.

Despite the considerable amount of connected devices, there is another constraint concerning the user-device experience. According to [12], there are three main categories: (1) Sequential Screening, (2) Simultaneous Screening and (3) Collaboration Scenario.

Until today, despite all the available technologies, we are not able to obtain a simple data-flow without starting from zero each time. An example is while a person leaves a particular place, she/he decides to begin writing a message on a mobile device, but enters the car, so continues the action using speech resources through the car, then, when arriving at the final destination finishes the message on a laptop, desktop or a smart TV.

Currently, there is no fluid exchange of information through all platforms, connected devices, and so forth. Protocols, messages, and certificate standardization to achieve full data-flow through connected devices are still lacking.

3 CUBE Model Proposal

It is a fact that multiple devices are a growing trend. However, despite all efforts in the last years, much must still to be developed to deal with this issue in a synchronized way.

The Internet was built to allow for caching of across its infrastructure, which can be helpful in many aspects. For example, a subsequent request for the same page along the same network path may be satisfied by a cached representation [15]. However, when we think of 1 or N connected devices to a user who needs to access her/his email, all this caching becomes useless.

When we think about computer-to-computer systems, a cost regarding transactional atomicity and also scalability is present [9, 15]. Despite the efforts to build a system using Liquid Software principles [1, 8, 12], there is a lack of architecture or framework able to deal with all the involved requirements, principles and constraints. Thus, it is in this scenario where we propose our user-centric based CUBE model, Fig. 2. It is possible to describe the CUBE model as a combination of two circumscribed cubes. Inside, the INNER CUBE represents the devices and surrounds the user (in red). Outside, the OUTER CUBE represents the services which the user can access from any point in the INNER CUBE.

The model description follows this sequence: A Server (S_1) requests an ordinary service (S_o). It needs to wait for the sum of the response time ($\sum t$). The

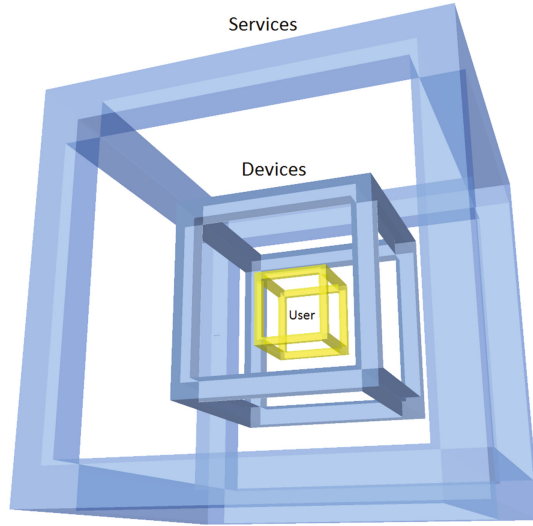


Fig. 2. The user is in the center surrounded by devices (inner CUBE) and services (outer CUBE).

time can be influenced by the sum of the distance to each service ($\sum d$), which can change according to the availability of routers. Once the Server (S_1) has received the service (S_o), it keeps all data as a cache. A second Server (S_2), which needs the same data once requested by (S_1), does not need to follow the entire path that (S_1) followed since all that (S_2) requires is the data.

After the first request from (S_2), if other Servers receive online requests of the same data from (S_1), all resources used in the initial (S_1) search are saved. With this feature, we also enhance energy consumption and throughput, among others. A comparison with a TCP-IP model, as displayed in the Fig. 3, can project how CUBE would act.

TCP/IP model	Protocols and services	Cubemodel
Application	HTTP, FTTP, Telnet, NTP, DHCP, PING	Connection
Transport	TCP, UDP	
Network	IP, ARP, ICMP, IGMP	
Network Interface	Ethernet	Network Interface

Fig. 3. All devices could use the same data after acquiring. from [20].

It is important to realize that each Server (S_1, S_2, \dots, S_n) is, in fact, a connected device owned by the same user. Since all the credentials, certifiers, headers and data in each layer of the TCP is already owned by the first Server (S_1), there is no need to perform all these steps again, since the owner is the same.

To activate any new device on its network, the user will need a trigger, such as a fingerprint or an encrypted password. By adding a device, we can combine two methods. First, the graph methodology [5] of level search, which can begin at any arbitrary node to all adjacent nodes. Second, using the Dijkstra algorithm [4] to search for the minimal distance from a node to another.

As for choosing a CUBE is to be as the adequate model for the approach, a Quadtree is applied [18]. The quadtree model is generally used to describe a class of hierarchical data structures whose common property based on the principle of recursive decomposition of space. According to [18], they can be differentiated by the following: (1) The type of data that they represent, (2) The guiding principle of the decomposition process, and (3) The resolution (variable or not).

Regarding the geometrical limitation of the CUBE, as an example, eight different devices using eight servers at the same time, two ways of dealing with this type of organization are available. The first is in the inner CUBE (devices), as the connection between vertices can provide a new device - volume approach, while the same principle can be applied to the outer CUBE (services). If technology limitations are present, another way to manage too many devices in the same user-network is duplicating the CUBE, which is possible using the Tridimensional approach of Archytas of Tarentum [3, 13].

We understand that the CUBE can be a model which could share restrictions or also act as a proxy to filter all Internet requests in the same network. Thus, brings forward the concept of domestic systems and their profiles sharing the same services but at different levels of a request.

As discussed, there are many current constraints which require attention, such as increasingly connected devices attempt to gain our attention. However, following the data-flow of these devices still require a significant effort to bring forward a unique platform.

However, to use the best practices, a perspective of the business process is required. While this approach deals with produced and consumed resources, a limited and linear vision of multi-connected systems is still prevalent. This scenario brings us to an edge which requires not depleting our resources, while, at the same time, providing all technical features.

4 Use Case Diagram

Due to its size and complexity, we will present the model from a theoretical perspective. However, it is possible to see how the data can be used and see fluidity through the devices.

The priority in this flowchart is to establish the *Layers Compliance*, as shown in Fig. 4. Thus, we can understand how we can have RESTful approach separate from other REST ones.

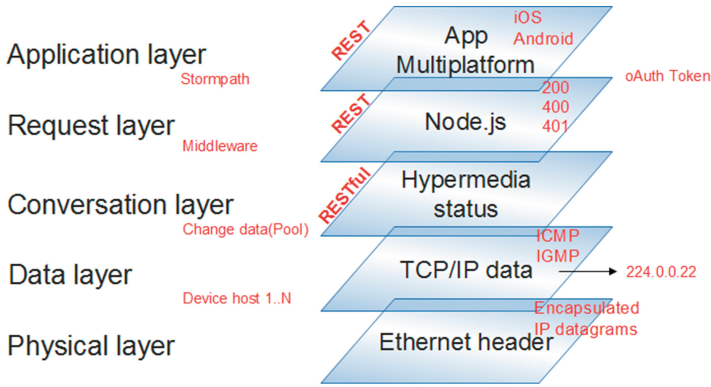


Fig. 4. Each layer implements a different task based on REST and RESTful platforms.

Flowchart readings in Fig. 5 begins with a *User* defining her/his first connected device to use it. The primary safety measure is passing through the process *Basic requirements*, where some procedures are executed, as to verify if there is enough energy in the device to start the process.

In the sequence, at the same time when the user chooses Device₁, another process called *Inner Cube* starts. Then it will create the *INNER CUBE*, which is responsible for keeping an updated list of hosts and also discovering all connected ones owned by the user.

Back to Device₁, for which the *Authentication Process* starts. Several steps will be running, among them, checking Internet connection availability. In this process, two *Layers* of the model are implemented, the *Application* and *Request Layers*, which are now able to deal with the users requests. Then, the next step can be either to synchronize the client and server sessions, (which is not explicitly drawn as a step in the flowchart in Fig. 5) or to start another process.

Meanwhile, the *OUTER CUBE* is created, once its subprocess was finished, it can be able to provide for the *User* a synchronized system of her/his use services: those available for any device of the user.

The automatic process *Saving Data* has the responsibility to retrieve data from the *Authentication Process* and the *OUTER CUBE*. We present here another *Layer Compliance*, the *Data Layer*, which, after treatment, will send data to the *INNER CUBE*.

One detail about the *Data Layer*, which for now is modeled in two perspectives. First, we adopt a level 2 Host extensions for IP Multicasting. This feature allows for a host to join and leave host groups, as to send IP datagrams to host groups [7]. In this scenario, our Device₁ will act as a host for all others in the *INNER CUBE*. To allow for one host to be the central node of a group, RFC 1112 [7] needs specific adaptations over ICMP and IGMP protocols for a range IP group address, e.g., 224.0.0.22. This approach optimizes the buffer required in the devices but imposes to be limited by the IP group address. However, still according to the RFC 1112, is possible to store a routing table in the host device.

FLOWCHART OF THE CUBE (SUMMARIZED)

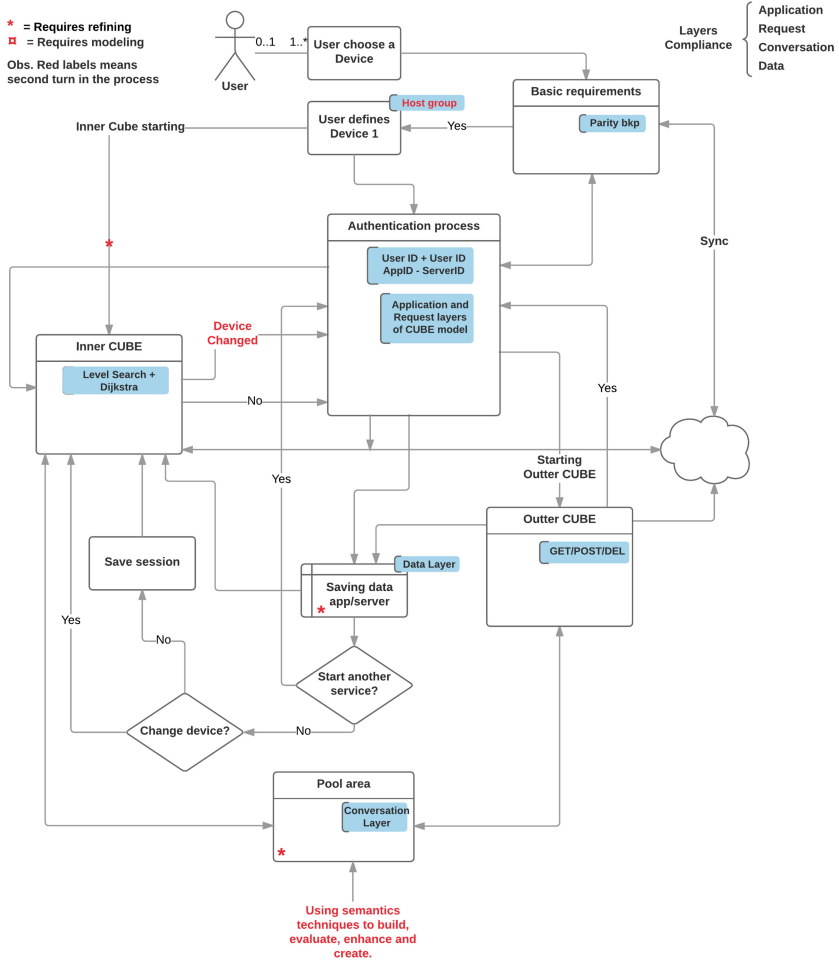


Fig. 5. How is the data-flow in the CUBE model when a service is started by a client.

For the second approach, we plan to use the P2P principals, based on the CHORD algorithm [21]. We can use the IP owned by the device and share the data across all devices in the INNER CUBE. However, this drives us forward two gaps, 1. Once the data is split over the user network, how we assure the packages assembled will reproduce it correctly and 2. Must we have copies of the P2P assembly on each device? For now, these gaps and also the Data layer still require our attention and modeling efforts to be aligned with the other Layers and produce the expected outcome.

The outputs provided by the previous process lead us to two different paths. First, if our answer is affirmative a service loop starts until the user chooses

to stop her/his activities, does not change the *Device* and call the process *Save Session*, which will send data to the *INNER CUBE*. Alternatively, for the second path, the user decides to change the device and continue to use her/his services in another place. In this case, *INNER CUBE* is also called to treat the user request sending forward the output provided.

Flowchart reading finishes with the *Conversation Layer*, the last one of the *Layers Compliance*. In our model, this layer has a unique behavior: to act as a pool area where all kinds of API can emerge, from Client or Server.

The pool will provide a place for the APIs changing data status of hypermedia formats. That is the main reason for this layer to be completely RESTful, as one of the level 3 RESTful characteristics relays in the ability of a service to change the set of links that are given to a client based on the current state of a resource [20].

5 Illustrative Scenario

This section describes how the CUBE model can run services across its structure, through a scenario of a multi-host environment for a Web-mail client. We illustrate this scenario in Fig. 6 and detail it hereafter. We would like to note that the system is still under development, and we expect some preliminary results to demonstrate its feasibility.

We must assume three conditions: 1. The user chooses a connected mobile as her/his first device and 2. She/he decides to run an email service and 3. The Car Operation Systems (COS) has a commanding voice available. Step 1 runs all internal procedures required for the authentication process and connection availability. Thus, in Step 2, the Application and Request layers, both REST, with StormPath and Node.js respectively, make the token creation and send the HTTP responses for the required service.

The user continues on the client mail. Then, the data-flow goes to Step 3, which is still under the assessment of the best methodology to use. However, this step is essential to define how the data will flow across the multi-host network.

Step 4 works under Semantic rules, based on the work presented in [19], where APIs can emerge both to client or server, to allow for the change in hypermedia status across the multi-platform. This step is also the Conversation Layer, built as fully RESTful. Moreover, all APIs that arrive at this layer is already secured by the token created in Step 1. Despite being shown as a linear model, the Conversation Layer is available at all times, allowing for data-change between devices and operational systems.

In Step 5, the user will move by car. Hosted by a COS, the CUBE will be displayed as an icon to retrieve all data/session/connections achieved previously herein. At this time, the available multi-hosts are synchronized, allowing a resume of any task, at any time.

The user arrives at her/his destination, leaving Step 5 with all synchronized data in her/his mobile until turning on her/his laptop, based on iOS, which

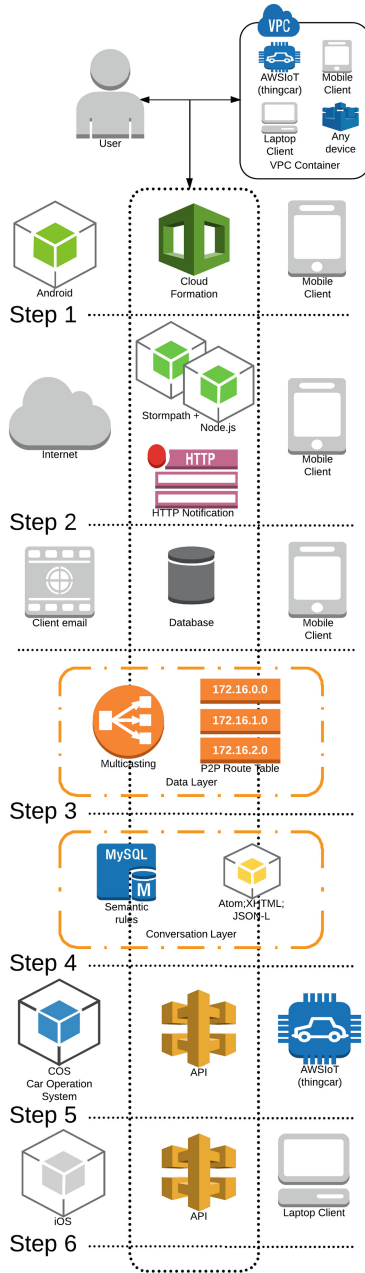


Fig. 6. The cube works as an ESB for the multi-platform environment.

will retrieve all data produced so far. Before finishing the process, an Internet connection is verified to allow the GET/POST/DEL methods to run.

Across this small scenario, we can tell that our contribution can start any service, local or not, no matter the chosen device. Thus, this kind of behavior is similar to those based on Enterprise Services BUS (ESB), once implements a software architecture for distributed computing. That means, whereas in general, any application using ESB can behave as server or client in turns, based on the EBS primary goal of an application integration of the heterogeneous and complex landscapes [6].

6 Multi-hosts Modeling API Implementation

The previous section has described how a service can flow through an environment with multiple connected devices. In this section, we present our CUBE model as an API, and how its modules work in our proposal to integrate both different devices and services on the same platform. An abstract perspective of the API modules is given in Fig. 7, which we will describe hereafter.

The content management system 101 includes two application portions 103 and 105, two applications program interface (API) portions 102 and 106, developed in JavaScript, and a database portion 104. Each portion provides a separate aspect of the content management system 101.

Starting from the external entity 100, the content management system, 101 will interact on two fronts, the user side and services environments. First, in the user environment 100, a device is specified as the first one. Its choice is made from a network of all connected devices available and owned by the same user.

In the Second, at the services environment 107, the CUBE will be able to receive and respond to different types of requests made by the user. Meanwhile, all those services will be synchronized in another part of the module, in the API 106. Moreover, under the resources saving perspective, this feature allows reducing consumption and throughput over the distributed infrastructure of the Internet.

These two external entities to the CUBE model interact directly with the two central concepts of our proposal. The *INNER CUBE* 102 and the *OUTER CUBE* 106, the two APIs for devices and services respectively. Thus, these both APIs are responsible for treating the different inputs and outputs to provide normalized responses for each interaction.

As next step, the *INNER CUBE* 102 has as main activity to interact with the user's devices 100, which can be changed during the service execution, and also treat requests in REST format, portion 103, through the layers Application and Request described previously.

Once the data has been treated and classified, access controls in the following layers set and make use of the metadata on the individual pieces of content. Users can see different data depending on the role that they are trying to accomplish in a specific device. Customized views into the data can be provided, and data integrity and minimal user confusion can be ensured.

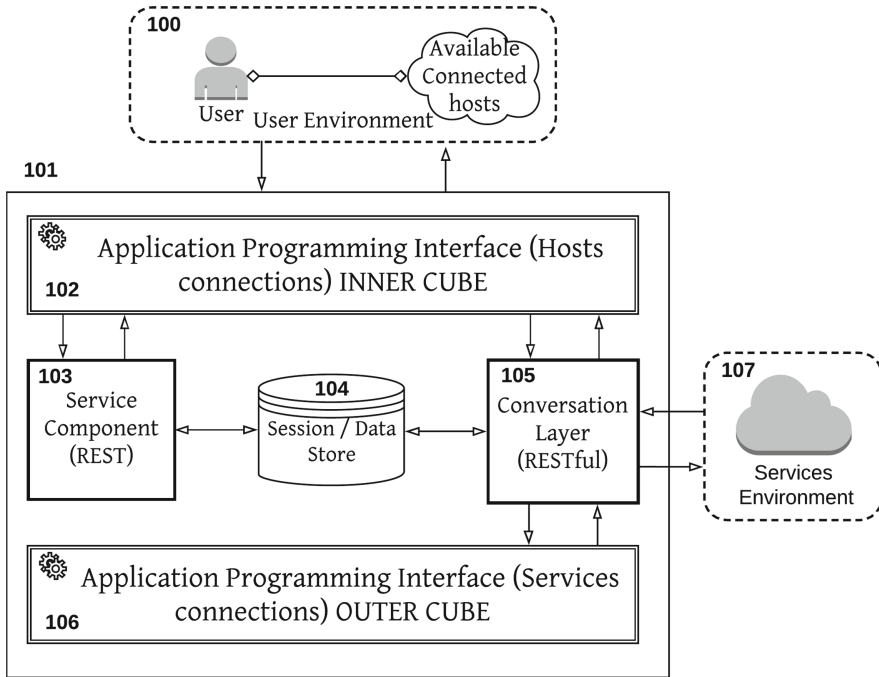


Fig. 7. The CUBE API works with multiple devices and services.

Workflows can be made to depend on content classification. For example, the image files may be shared inside multi-host environment, allowing their access to all system. In the opposite, an on-demand video must be redirected to each device in use, that means, changed by the user in her/his flow. Thus, this may require the use of complex classification-based workflows, therefore, automatically route for all connected and available devices.

While services and devices change, the information as sessions retrieved by the API 102, and also the data received can be stored in the database portion 104 via application module 103. Thus, when any change of device is required, the user does not need to start from scratch to retrieve or fill in her/his information.

The application portion 105, working as a pool area, is built under RESTful principles and is responsible for changing information across the multi-platform environment. Moreover, this part of the module provides a secure connection for the user devices through a token created in the *INNER CUBE* 102 by the join of User ID from device's and service which are provided by the *OUTER CUBE* 106. This type of addressability through a resource identification allows the pool to ensure a stateless interaction between services.

While services and data are requested by the *OUTER CUBE* 106, the *conversation layer*, portion 105, connects with the services environment 107. Its connection and data are stored in the *database* 104. Moreover, this feature gives the flexibility and fluidity required in the Liquid Software approach.

We notice that this kind of long description to access the APIs is not linear. That means, once all the connected devices are identified, they will be available for the user changing according to her/his needs. Thus, the APIs will be available for all devices. Moreover, it will be possible to resume the running activity in any device still available. For this purpose, the API will be displayed as an icon and synchronized among all the devices.

7 Conclusions

Until today, work with the same services or applications in different devices is still a challenge which requires more investigation. Moreover, there is still a gap to achieve both a multi-host and services synchronization over multi-platforms, described herein. Furthermore, we have presented our CUBE Model under the principles of Liquid Software, which should be a trend in a world which is more and more connected each day.

Throughout an User-Centric approach and combining principles of REST and RESTful model, our proposition, is a model able to converge multiple and heterogeneous environments with different behavior and complex systems. Thus, it displays the ability required in dealing with all the challenges of the Internet platform in a multi-host environment.

To achieve fluidity, the CUBE Model applies the best practices of different technologies described herein. Moreover, the model aims to fulfill the principals and constraints of these technologies regarding the issues of security, stateless interactions and providing a uniform interface shared by all resources, across separated layers for the REST and RESTful applications.

Our future steps drive us for the specification of the Data Layer regarding our objectives. Once defined, we can attach this Layer to our model and test it under the principles of Liquid Software. The last step stands in establishing the basic semantic rules to run in the Conversation Layer to achieve level 3 Maturity and allow for a change status in hypermedia data. After these results, we will model the CUBE as an API and evaluate it. Besides, we will assess whether the CUBE will be a model for a framework or an application able to deal with the increasing requests of connected devices.

References

1. Gallidabino, A., Pautasso, C., Ilvonen, V., Mikkonen, T., Systä, K., Voutilainen, J.P., Taivalsaari, A.: On the architecture of liquid software: technology alternatives and design space. In: Proceedings of 13th Working IEEE/IFIP Conference on Software Architecture (WICSA). IEEE (2016)
2. Bennara, M., Mrissa, M., Amghar, Y.: An approach for composing restful linked services on the web. In: Proceedings of the 23rd International Conference on World Wide Web, WWW 2014 Companion, pp. 977–982. ACM, New York (2014). <https://doi.org/10.1145/2567948.2579222>
3. Boyer, C., Merzbach, U.: A History of Mathematics. Wiley (2011). <https://books.google.fr/books?id=bR9HAAAAQBAJ>
4. Brandes, U.: A faster algorithm for betweenness centrality. *J. Math. Sociol.* **25**(2), 163–177 (2001). <https://doi.org/10.1080/0022250X.2001.9990249>
5. Bulitko, V., Lustrek, M., Schaeffer, J., Bjornsson, Y., Sigmundarson, S.: Dynamic control in real-time heuristic search. *J. Artif. Intell. Res.* **32**, 419–452 (2008)
6. Chappell, D.: Enterprise Service Bus. O'Reilly Media Inc., Sebastopol (2004)
7. Deering, S.: Host extensions for IP multicasting. RFC 1112, RFC Editor, August 1989. <https://www.rfc-editor.org/info/rfc1112>
8. Hartman, J.J., Bigot, P.A., Bridges, P., Montz, B., Piltz, R., Spatscheck, O., Proebsting, T.A., Peterson, L.L., Bavier, A.: Joust: a platform for liquid software. *Computer* **32**(4), 50–56 (1999)
9. Haupt, F., Leymann, F., Pautasso, C.: A conversation based approach for modeling REST APIs. In: Proceedings of the 2015 12th Working IEEE/IFIP Conference on Software Architecture, WICSA 2015, pp. 165–174. IEEE Computer Society, Washington, DC, USA (2015). <https://doi.org/10.1109/WICSA.2015.20>
10. John, D., Rajasree, M.S.: RESTDoc: describe, discover and compose restful semantic web services using annotated documentations. *Semant. Technol.* **4**(1), 37–49 (2013). <http://airccse.org/journal/ijwest/papers/4113ijwest03.pdf>
11. Lanthaler, M., Gütl, C.: A semantic description language for restful data services to combat semaphobia. In: 5th IEEE International Conference on Digital Ecosystems and Technologies, IEEE DEST 2011, pp. 47–53, May 2011
12. Mikkonen, T., Systä, K., Pautasso, C.: Towards liquid web applications. In: Cimiano, P., Frasincar, F., Houben, G.-J., Schwabe, D. (eds.) ICWE 2015. LNCS, vol. 9114, pp. 134–143. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19890-3_10
13. O'Connor, J., Robertson, E.F.: Archytas of Tarentum (1999). <http://www-history.mcs.st-andrews.ac.uk/Biographies/Archytas.html>. Accessed 24 Jan 2017
14. Panziera, L., De Paoli, F.: A framework for self-descriptive restful services. In: Proceedings of the 22nd International Conference on World Wide Web, WWW 2013 Companion, pp. 1407–1414. ACM, New York (2013). <https://doi.org/10.1145/2487788.2488183>
15. Parastatidis, S., Webber, J., Silveira, G., Robinson, I.S.: The role of hypermedia in distributed system development. In: Proceedings of the First International Workshop on RESTful Design, WS-REST 2010, pp. 16–22. ACM, New York (2010). <https://doi.org/10.1145/1798354.1798379>
16. Pautasso, C.: RESTful web services: principles, patterns, emerging technologies. In: Bouguettaya, A., Sheng, Q., Daniel, F. (eds.) Web Services Foundations, pp. 31–51. Springer, New York (2014). https://doi.org/10.1007/978-1-4614-7518-7_2
17. Richardson, L., Ruby, S.: Restful Web Services, 1st edn. O'Reilly, sebastopol (2007)

18. Samet, H.: An overview of quadtrees, octrees and related hierarchical data structures. *Theoretical Foundations of Computer Graphics and CAD*, 1st edn. Springer, Heidelberg (1988)
19. da Silva, C.P.: Text2MARK: a text mining tool to aid knowledge representation - (MARK2). In: 2014 14th International Conference on Intelligent Systems Design and Applications, pp. 199–204, November 2014
20. da Silva, C.P., Messai, N., Sam, Y., Devogele, T.: Diamond - a cube model proposal based on a centric architecture approach to enhance liquid software model approaches. In: *Proceedings of the 13th International Conference on Web Information Systems and Technologies - Volume 1, WEBIST*, pp. 382–387. INSTICC, ScitePress (2017)
21. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.* **31**(4), 149–160 (2001)
22. Trinh, T.D., Wetz, P., Do, B.L., Kiesling, E., Tjoa, A.M.: Semantic mashup composition from natural language expressions: preliminary results. In: *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services, iiWAS 2015*, pp. 44:1–44:9. ACM, New York (2015). <https://doi.org/10.1145/2837185.2837194>
23. Vinoski, S.: Serendipitous reuse. *IEEE Internet Comput.* **12**, 84–87 (2008)



Harnessing Community Knowledge in Heterogeneous Rule Engines

Kennedy Kambona^(✉), Thierry Renaux, and Wolfgang De Meuter

Software Languages Lab, Vrije Universiteit Brussel,
Pleinlaan 2, 1050 Brussels, Belgium
{kkambona, trenaux, wdemeuter}@soft.vub.ac.be
<http://soft.vub.ac.be>

Abstract. Currently, there is a lack of rule-based approaches that offer rich semantics that developers can use to exploit community knowledge contributed by distributed heterogeneous clients. Such abstractions can be useful in a number of applications to deal with the problem of orchestrating data patterns in a heterogeneous setting. This work presents scope-based reasoning in heterogeneous rule engines as a means to capture collective intelligence via community knowledge. Using scoped rules, rule designers can detect patterns in real-time data and to realise grouping structures in heterogeneous applications backed by a common rule-based system. The proposed solution exploits the fact that much of the heterogeneous community knowledge significant when performing reasoning and deductions can be structured hierarchically. We evaluate our work through a simulated case study, confirming that our technique presents a viable approach for efficiently processing community knowledge in heterogeneous environments.

Keywords: Rule-based systems · Community knowledge · Rete Scopes · Business rules

1 Introduction

As a result of today's dynamic software environment, clients require real-time processing in modern data-intensive applications: where the clients contribute data as events and expect to receive instantaneous feedback through notifications. Events can consist of both low-level data, such as raw GPS coordinates, and high-level data that other applications depend on, such as detecting a package that is late for delivery. Such event data can be sent continuously over distributed networks to application servers in various forms as out-of-order, partially unbounded and/or time-varying sequences. Accordingly, there is a current need for *knowledge-intensive* techniques that ease the dynamic definition of constraints in order to extract value from such continuous, *reactive* event data. This will enable the system to infer potential higher-level knowledge that contributed data may uncover. Currently, the technologies available to meet these

goals is often times complicated and limiting due to the non-determinism and sheer volume of data to discern patterns on [1].

Recent advancements in tackling this problem has re-discovered the use of forward-chaining rule-based systems (RBS) in areas that embrace the use of *business rules* [2]. Rule-based systems provide a declarative approach that represents the conceptual logic of a system in an application-independent way. The fundamental power that rule-based systems expose is not only from complex inference mechanisms but also from the ability for them to embrace rich knowledge bases that reflect aspects of the real world [3].

Fundamentally, rule engines were designed in the era where isolated computing was prevalent. At the time, rule engines were programmed to encode a localised set of rules and to work on homogeneous data. In contrast, the online software ecosystem is definitively characterised by a heterogeneous environment where different types of client devices contribute vastly diverse types of data for processing. Sharing brought about by Utility Computing [4] is significant in exploiting the *collective knowledge* that can be discovered from the data that these devices contribute.

Sharing however unearths issues that can be attributed to classically-isolated rule engine design. Classical rule engines suffer from a lack of proper modularisation when installed to serve heterogeneous settings with shared data: they are characterised by a flat design space where activations could be observed from all data without discriminating their sources. They are thus said to be *non-reentrant* [1]. Most rule-based approaches provide basic techniques that do not intrinsically support the flexibility and expressiveness in customising specific client behaviour. The methods employed to mitigate these problems unnecessarily retract the gains made by using a rule-based system in the first place: they increase the complexity of application development making it fallible, and muddles the design of the conceptual logic of the application. They furthermore lack the proper mechanisms in which to exploit improvements in efficiency that can be realised in a heterogeneous setup.

In this work we augment the support the efficient processing of heterogeneous, multi-user applications through a technique that enforces the consolidation and partitioning of client constraints and data defined using special rule-based programming constructs. We present *scoped rules*, that enable rule creators to distinguish between events pertaining to different sources while keeping this logic cleanly separated from the application logic. As such, the basic purpose of the rule is not muddled with the logic required for distinguishing clients, leaving the logical intent of a rule easy to understand for a rule creator.

At the same time, scoping enables the system to exploit a number of performance optimisations in the server's rule engine during its matching process. The approach of encoding the physical, structural or other logical organisations of multi-user applications eases the computational workload of the inference algorithm. We show that the underlying rule engine can effectively use such abstractions to generally decrease the engine's overall response time and thus improving overall processing efficiency when compared to classic methods.

The document is organised as follows. Section 1.1 discusses the significance of incorporating community knowledge and presents a motivating example of a representative heterogeneous application. Sections 2–5 describe the proposed heterogeneous rule-based system and its execution semantics. Section 6 then proceeds to evaluate the work against current methods, and Sect. 7 outlines the ways in which similar approaches can be used to support the capture of community knowledge in other heterogeneous environments. Section 8 recapitulates the main points and points out some limitations and possible avenues for future improvements.

1.1 The Significance of Community Knowledge

Value is a significant feature when reasoning on data contributed by client devices to discern useful patterns [4]. Discovering interesting patterns from such intermittent, dispersed or entangled pieces of data greatly improves data quality. A good analogy for this is the ancient *Parable of the Blind Men and the Elephant* that originated from the *Rigveda* collection from ancient India [5]. The fable describes a number of blind men sizing up an unknown object, a large elephant. Their goal is to try and determine what the object or creature is, and project the result to the others. Each man feels only one part of the elephant’s body and are then required to describe the elephant based on this. Since each man has only a limited, *local* perspective of the elephant, they come up with different conclusions of what the object can be: for instance, each says the elephant could be a wall, a spear or a rope depending on the limited region (respectively, the side, the tusk or the tail) they can access, Fig. 1. The fable ends with the blind men in complete disagreement of what the object is.

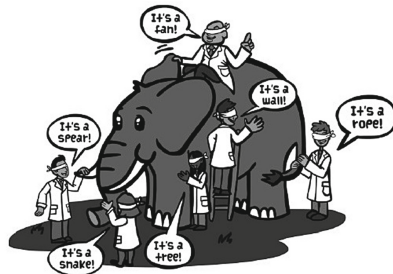


Fig. 1. The Blind Men and the Elephant – Each blind man has a limited view of the elephant and come up with different conclusions of what the object is (Image sourced from [6]).

In the same way, value from data contributed by a variety of sources or clients can potentially be increased if it is processed collectively (cf., discovering that the object is indeed an elephant). This will significantly increase its value

because the information is fed into a deterministic process where patterns within a set of collective information can be ascertained. One field where this manner of processing is of practical significance is in the organisational intelligence domain, where the result is often termed as collective intelligence [7].

This work goes further in defining a concise collection of useful information sourced from different clients that can be grouped according to logical or physical structures, that we denominate as **community knowledge**. Community knowledge is important in the context of heterogeneous systems due to the vastness and diversity of the types of information that can be produced, collected and processed from different clients. This type of composite knowledge encompasses the variety of data that is often contributed in these environments.

This outlook stands above other similar approaches because a diverse range of autonomous sources can contribute reactive data in one integrated system. This has a profound effect on its complexity and the relationships that can be unearthed. For example, traditional systems were focused on relating features with a single independent entity, such as age, date of birth, gender, etc. Today, an individual's features as well as relationships form a social connection with other entities through relatable hobbies and other interests. In traditional systems, individuals are only linked if they have same basic features. In a modern, dynamic community, two individuals can be linked via these social connection relationships even though they do not share any of these intrinsic features.

1.2 Motivating Example

We now present a representative practical example that motivates the need for exploiting community knowledge in a heterogeneous environment. To highlight the requirements that such a system should meet, we describe a scenario of a security monitoring system in an university environment that monitors access patterns. The service can be deployed in a heterogeneous configuration (such as through a Cloud Access Security Broker [8]) to monitor and log access in several institutions. The example follows that explained in [1].

University Services Access Control. The security departments of universities in Brussels have embarked on improving the security of their campuses. They have passed regulations that require the security departments enforce protocols to monitor accesses of all staff and other students and staff in their institutions.

Devices that scan badges issued to students and staff have been installed at major access points throughout the university premises. The badges are contactless smart cards that can be read by the access devices. To gain access to any part of the university, a person is required to scan their badge on the device. We outline some of the security protocols that have been drafted by the security department from the regulations below.

1. Students at all levels (bachelor, master, etc.) have access to classrooms during class times on weekdays
2. Only vehicles of students and staff are allowed to enter the underground parking in campus buildings

In Fig. 2 we illustrate a simplified common structure for a university consisting of different hierarchies: personnel, physical structures and research department hierarchies. As a result, specific departments and units can therefore define custom access policies:

3. Biology department students are allowed access to all labs in the (sub)departments in the weekends if accompanied by senior academic staff
4. Only campus bank employees and consultants have access to the bank back office during working hours

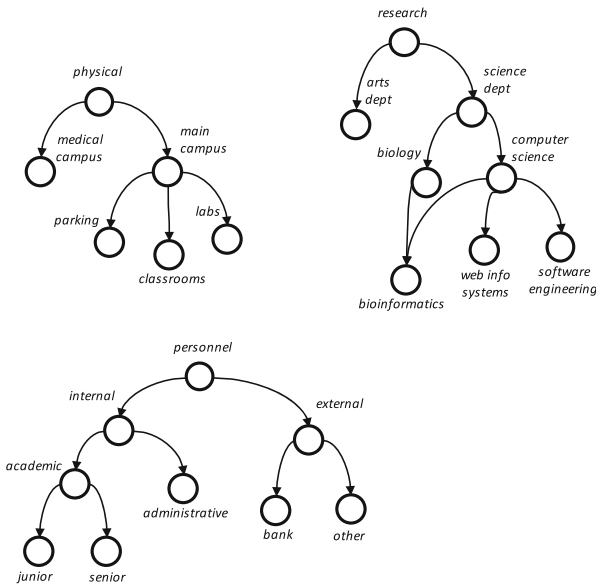


Fig. 2. Structural organisation of a university [1] – The structures can be modelled as physical locations, research groups and personnel; spanning students, staff and physical structures.

A system supporting this example should be capable of defining both types of policies, and of processing the requests from all users efficiently against the protocols at varied times whenever any request is made. For instance in policy 1, when a student on a university accesses a classroom during class times the monitoring dashboard would show a status to indicate whether the access is acceptable or otherwise. Such kind of processing model is common in forward-chaining rule-based systems in the complex event processing domain.

2 The Serena Rule-Based System

The example outlined in the previous section is a representative of a reactive heterogeneous application. In this work, we particularly target the dynamic design of such knowledge-intensive, data-driven applications that continuously stream data back and forth between clients and the server. These systems are required to manage the shared knowledge base reused by the various applications they support. In order to reason about data sent by client devices and to extract higher-level knowledge from it, it is vital that the value of the sent data be processed efficiently. Instead of hard-coding all this shared knowledge using conventional techniques, developers often encode this knowledge in the form of rules to specify detection logic. In such situations, a modern rule engine can be used to accommodate the knowledge for all clients of a heterogeneous configuration. This is the main idea behind the design of the Serena framework [1].

2.1 The Serena Framework

Serena is a rule-based framework that augments an event-driven web server with a forward-chaining inference engine that processes event data reactively using rules. In Serena clients create and install the logic reactive rules that define the complex events they are interested in. The rules specify which data to match, and once activated the rule can send this activation as notification to clients.

2.2 Serena's Execution Semantics

Producing efficient rule-based functionality requires maximum efficiency due to complex pattern-matching techniques of rule-based semantics. Reducing the amount of matching in rules therefore guarantees faster server execution. The Serena runtime is based on the production systems model of knowledge representation [9], which uses data-sensitive rules rather than sequenced instructions as the basis of computation.

We use the university access control's example protocol 1 to explain the semantics of execution in a typical heterogeneous rule engine.

Rule Syntax. Rules support defining constraints that will enforce pattern-matching within conditions in order to execute some actions. The university policies from the scenario in Sect. 1.2 can be easily expressed in a rule-based format. We show such a rule to be added by a university security staff using a syntax similar to JSON Rules [10] in Listing 1.1 for the classroom protocol 1. Remember that the protocol specified that students are only allowed access to classrooms on weekdays. The rule in JSON format is received by the framework's server when shipped from the client.

Listing 1.1. Rule for allowing classroom accesses to a student.

```

1  {rulename: "classtime-access1",
2  conditions:[
3  {type:"student", name: "?name"},
4  {type:"accessdevice", name:"?dev", location:"classroom"},
5  {type:"accessreq", id: "?reqid", person: "?name", time: "?t", device: "?dev"},
6  {type:"$test", expr:"(hourBetween(?t, 8, 20) && (isWeekday(?t) == true) )"}
7  ],
8  actions:[
9  {assert: {type: "accessrep", reqid:"?reqid", allowed: true}}
10 ]
11 }

```

The rule consists of a name, the left-hand side (LHS) and the right-hand side (RHS). The rulename identifies the rule. The LHS contains conditions (lines 2–6) that specify constraints on incoming events for event detection. The RHS contains the actions to be taken when the conditions have been fulfilled (lines 8–10). The LHS of the listed rule definition captures the access request from a person on an ID scanning device within the specified time periods (line 6). In the rule the ‘?’ operator denotes a variable binding (e.g. ?name in lines 3 & 5).

When all the conditions specified in the LHS are satisfied, then the actions defined in the RHS are activated. In the example rule, the RHS asserts that the access request has been granted by the reply in line 9 from the request captured in line 5.

The Rete Algorithm. When the rule engine receives the rule it builds a discrimination network via the Rete algorithm [11] in its inference engine. Rete compiles rules into a data-flow graph that filters facts (data) as they propagate through nodes performing a match-select-execute process. Rete inference engines perform efficient matching, a technique that reasons over the data to detect consistent bindings for constraints in rules that need to be fulfilled. Efficient matching is achieved by exploiting two characteristics. The first is structural similarity which involves sharing similar nodes when building the graph. The second is temporal redundancy which is the caching of intermediate matched data tokens between cycles of incoming results, albeit at the price of higher memory usage from the added caches.

The Rete Graph. We show the graph for the *classtime-access* rule previously shown in Listing 1.1 after addition in the server. The graph consists of two regions, the alpha and beta network. The alpha network contains alpha nodes that perform intra-condition tests, such as the leftmost alpha node that checks if a fact is of type `student`.

The beta network is built in the lexical order of the conditions in a rule, forming a left-associative binary tree. Two-input *beta nodes* or *join nodes* perform tests or joins between conditions on their left and right inputs. Each beta node is associated with its *beta memory* and holds the intermediate join test results. The leftmost beta node in Fig. 3 from [1] performs joins for a the name of a students with the name of the person performing the access request.

If the test passes, it will create a *token* of both facts in the result and send it to the next node. Each beta node can be connected to subsequent nodes in the beta network as a left input. In most beta nodes the right input is connected to the output of an alpha node's memory. In Fig. 3, the second beta node receives the token and performs joins of facts from a scanning *accessdevice* with the device of the *accessrequest* made.

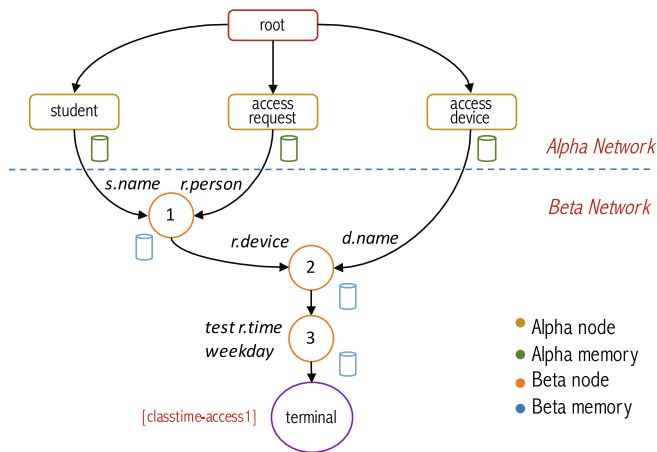


Fig. 3. The Rete graph built for the classtime access rule – The graph contains alpha and beta nodes with intermediate memories [1].

The beta network also hosts the nodes that represent test conditions as beta test nodes, e.g., the third beta node in Fig. 3 that checks for the compatibility of the time that the access request was made. The last beta nodes in any Rete graph represent the full activation of a particular rule (or in some cases, rules) and is named a *terminal node*. If a token reaches a terminal node then the rule associated with that node, in this case the *classtime access* rule, will be instantiated.

The Matching Process. In Serena, every fact base update triggers a matching process. The matching process searches for consistent bindings between facts and the existing rules. Incoming data is received by the server which it creates an instances of facts. The facts are then inserted into the graph from the root node. Facts traverse down the network as they are processed and forwarded by nodes, which store intermediate computations in local memories.

If a number of students arrive at the university by accessing campus entry points, this data can be captured and inserted as facts into the rule engine. The student facts will be inserted into the network starting from the root node and will eventually be stored as the intermediate results in the *student* node. Similarly, the access devices that are online will be stored in the memory of the

accessdevice node. The beta node 1 now will perform its join operations when an access request fact is received.

When a student requests access to a classroom at around 12pm (this request should be granted), the engine will receive it and eventually add it as an accessrequest fact to the Rete graph. The fact will be sent from the root to the accessrequest node. This node will store the fact in its alpha memory and will send it to its child, beta node 1.

It will be received at the right input beta node 1, causing a *right activation* which will issue a request for all the items in its left parent to compute consistent bindings for the fact. On the other hand, a *left activation* is triggered when a data item is received at the left input: where the join test will request all items from the node's right parent (which is always an alpha memory).

The beta node 1 will therefore request items from its left parent the student alpha node. The alpha node will send all student facts that it contains. Beta node 1 now proceeds to perform its join test (`s.name == r.person`), which checks if the name of any of the student facts is the same as that of the accessrequest fact.

When the person that made the request matches the name, then node first creates a new token by appending the student fact with the access request fact, stores this intermediate result in its beta memory, and sends the new token to its child, node 2. Thereafter, the same sequence of steps occur at node 2, but this time a left activation is triggered to find out compatible access devices by performing join tests (`r.device == d.name`) on all devices from the alpha memory of accessdevice.

If a compatible device is found then a token is created and sent to the test node 3 that checks whether the time for the request is within 8 am and 8 pm. The time 12 pm succeeds the test, so the token finally reaches the terminal node, which means that the rule should be activated. In this case, the request made by a student to enter the classroom will be granted, and this rule and its bindings are sent to the scheduler for execution and eventual notification to the client.

The Cost of Matching. The matching stage determines the rules that are relevant to the current state of the fact base for activation. As identified in [12], a major bottleneck in the Rete algorithm is, unsurprisingly, the expensive computations during this stage. Concretely, as much as 90% of the execution of a Rete-based system can be spent in the match phase, with the number of join comparisons made dominating the time the matching process takes. For this reason, the main area of improvement when looking for avenues to speed up any Rete-based rule engine execution is join computations during the matching process.

3 Reentrancy in Heterogeneous Rule-Based Systems

Reentrancy is a phenomenon used to describe programs written in such a way that the same copy in memory can be shared by multiple users effectively. A program is reentrant if distinct executions of the program on distinct inputs cannot

affect each other, whether run sequentially or concurrently [13]. Reentrant code is a requirement in common multi-user systems such as operating systems, where system programmers ensure that whenever a program is executed for a particular user there can be no other instructions that can modify data intended for another user. This way if the program is interrupted due to scheduling optimisations, for example, the program can be *re-entered* at any point in time without concern that programs that were executing during the interruption modified any of its data¹.

Rule engines were not conceptually designed to work in the heterogeneous environment. This is because rule-based systems are characterised by a uniform design space where a number of unordered rules are referencing a global working memory. When ported to heterogeneous environments such as in the multi-tenancy [1] context, these rule-based systems are revealed to be intrinsically non-reentrant where in this flat design space activations could be observed from all asserted facts without discriminating their sources. We exemplify the problem next using the university access control example.

3.1 Example: Non-reentrancy in Classical Rule-Based Systems

Now the security team of the university, *University1*, has installed their rule in the Serena server running a heterogeneous RBS. The team from the second university, *University2*, designs several rules using protocols extracted from the same security regulations. We observe the situation when they proceed to naïvely upload their similar classtime access rule.

In Rete rules are technically shared in their entirety within the network. As mentioned, structural similarity promotes sharing of nodes performing the same test but corresponding to different rules. When a security person from *University2* adds their classtime access rule *classtime-access2*, this results in the Rete graph that is structurally the same as before the addition of the rule – however this time the terminal node is tagged with activation of both rules. Because the terminal node was tagged with both rules, when a student from either university makes an access request in a classroom both rules will be activated on both clients, *University1* and *University2*. This can indeed be an undesirable result in heterogeneous setups, since now both companies can have notifications of granted accesses from unknown parties on their dashboards or in their system logs.

This simple example exposes the fact that in order to fully exploit capturing community knowledge in a rule-based system operating in a heterogeneous setup, it is vital that the system should avail mechanisms in which problems brought forth due to lack of reentrancy be suitably addressed.

Classic rule-based systems are thus said to be fundamentally non-reentrant. Given varied data sources, rules intended for one specific source or a number of

¹ Reentrancy as used here relates to the notion of reentrant procedures in multiuser systems programming and excludes those related to concurrent access and recursive method calls.

sources can be activated by data from other sources. Therefore, multiple heterogeneous data sources can lead to unexpected behaviour during execution cycles of the rule engine. One undesirable consequence is that rule activations can be observed from all asserted facts without discriminating their specific sources. In effect, the difficulty in localising rule control makes it hard to orchestrate the behaviour of rules in these settings.

3.2 Common Workarounds in Classical Rule-Based Systems

To fully exploit community knowledge in rule-based systems within heterogeneous contexts, there is need to solve the lack of reentrancy brought about by sharing in the Rete network. In [1] we describe traditional techniques applied by developers of rule-based systems to enforce discrimination during execution in a Rete graph. The techniques included using relation facts to introduce situational state in the execution cycle and test expressions to perform beta tests on tokens using discriminatory test conditions in rules.

The two approaches have similar limitations. It is generally undesirable to have rule condition logic (or application logic) interspersed with event source identification as noted in [14] in the context of notifications in event-based systems. This is mainly because they pollute the logical intent of the rule making it unnecessarily complex.

In more complex rules, it becomes tedious to distinguish which conditions need to be infused with the information that identifies clients and which ones do not. Using ad-hoc methods forces rule designers to hard-code distinctions between clients and their data sources, and quickly becomes complex and fallible as the number of clients and the relationships between them increase; or when the relationships become complicated to enforce using rule semantics. In a heterogeneous setup, failure to properly make these distinctions can also cause unintended rule activations to leak in other clients.

In summary, these classical methods are problematic because they (1) draw context knowledge into application components that relates to the interaction with outside entities rather than the rule implementation (2) pollute the logical intent of the rule designer (3) complicate rule implementation and makes the process fallible (4) in some cases impact the underlying Rete graph by creating additional nodes requiring more computations.

3.3 The Role of Client Relationships in Community Knowledge

Rule complexity quickly becomes more tedious in heterogeneous configurations. In this section we show the significance of relationships in exploiting community knowledge, especially in situations that contain more complex internal structures.

Usually, heterogeneous setups contain complex structures that are modelled according to physical or logical *relationships* among participating clients. These relationships can be based on aspects such as the principle of locality among interacting components in event-based systems [14] and encapsulation

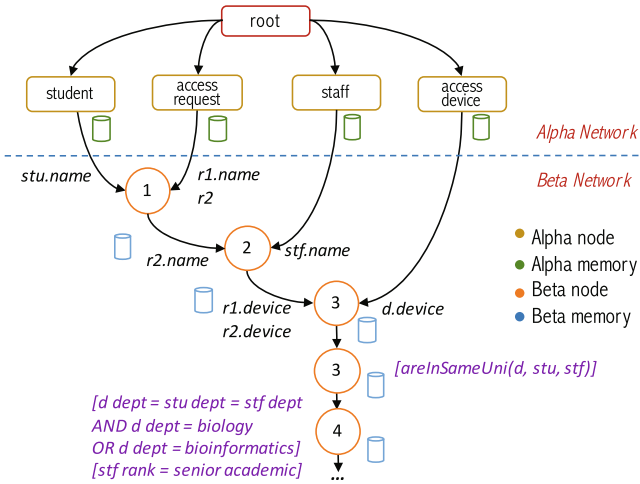


Fig. 4. Rete graph for policy 3 with additional tests for department-level checks [1].

in object-oriented systems. In simple situations these relationships can be based on the practical application-specific semantics tied to underlying structures, operations and processes of clients. For instance, research groups can belong to (sub)departments, hobbies can be categorised into hierarchies of interest groups and sensor area zones can be contained in levels of administrative units.

Consider the rule from protocol 3 that specified that students from a department in the university can have special access times to their (sub-)departmental labs. The rule developed for the protocol is shown in Listing 1.2. There are two access requests in lines 7 & 8 from the student and the senior academic staff, so we need to check if they come from the same department and if the department is biology or bioinformatics (line 9) as per the policy and the defined structure in Fig. 2 (note that the rule is more complex if the student and academic come from different departments). The resulting Rete graph for policy 3 with test expressions is shown in Fig. 4. Defining rules for such situations increases the problems brought about by a lack of reentrancy as seen in the expression in line 9, which tries to capture data from the same biology department or its sub-department bioinformatics. In the graph, node 4 contains further tests that implement the above constraints.

Listing 1.2. Rule for biology dept. weekend lab access.

```

1 {rulename: "biology_weekend_access",
2   conditions: [
3     {$stu: {type:"student", name: "?stuname"}},
4     {$stf: {type:"staff", name: "?stfname"}},
5     {$d: {type:"accessdevice", name: "?dev", location:"labs"}},
6     {type:"accessreq", person: "?stuname", device: "?dev"},
7     {type:"accessreq", person: "?stfname", device: "?dev"},
8     {type:"$test", expr:"( areInSameUni($stu.dept,$stf.dept,$d.dept) )"}
9     {type:"$test", expr:"( ($stu.dept == $stf.dept) && ($stf.dept == $d.dept) &&
10      ← ($d.dept == 'biology' || $d.dept == 'bioinformatics') )"}
11   ]
12 }
```

4 Requirements for Heterogeneous Rule-Based Systems

This section outlines the requirements of heterogeneous RBSes to suitably provide flexible mechanisms that can be used to exploit capturing community knowledge.

4.1 Metadata Model for Managing Client Data

Heterogeneous RBSes require an approach that imposes a uniform and consistent model supporting the identification of clients as event sources thus enabling the mapping of data items (and rules) from different contexts to specific client(s). This in effect will form the basis for the underlying rule engine to be able to ascertain the sources of constraints and events; using this to determine their respective execution contexts. The proposed approach should use a metadata architecture that is application-agnostic in order to promote the normal semantics of a rule-based system thus absolving the end-user application from the nuances that would exist without the model. On the server side, the approach should also be able to reason about the data model with the least effect on the underlying processing cycle of the rule engine.

4.2 Formalised Model for Grouping Clients

One of the challenges that heterogeneous RBSes face is ways in which to partition execution contexts, brought about as a direct result of how client rules are shared within the Rete graph generated by the inference engine for efficiency purposes. Many heterogeneous environments with relationships form communities through some form of *grouping* [15]. A heterogeneous RBS can use this aspect to provide an extensible model that captures the structures of clients and describes any possible compositions dynamically. In addition, this approach can be effectively captured in a more flexible way through a formalism that is based on the aforementioned metadata definitions. The formal model should be designed around a hierarchical structure that lends itself well in progressively describing the relationships between identified application-dependent classifications, useful in different real-world scenarios.

4.3 Execution Model for Selective Computations

During execution, rule engines do not provide selective matching by imposing any discrimination in rule execution: all rules are under consideration in a match cycle. Indeed, the diversity of data sources in a heterogeneous setup requires further precision in the execution context. Even the selection strategy of rule activation never actually depends on selective rule matching, but in reality is based on the recency of an instantiation and requires special ordering of rules. An extensible formalism can be exploited to effectively perform selective execution of the rule engine by discriminating or consolidating the data residing in the

heterogeneous system. These computations are those that try to perform the “*are the tokens that we want to match originating from the same source?*” - check that is needed to find consistent bindings for the different data sources, to avoid unintended activations with data from unwanted sources. The implication therefore is that the internal structures of clients should be reflected in the runtime in order for it to efficiently process the requests within the confines of each client’s configuration or constraints. Implementing these proposals will result in the rule engine performing selective execution of rules thereby, in a number of cases, reducing the amount of computations performed by the engine during its execution cycle.

4.4 Flexible Model for Notification Semantics

In such heterogeneous contexts, a notification is a message sent to a specific client(s) that reifies an event resulting from a rule activation. The notification carries the data that accompanies the activation, but may also contain additional metadata such as the time of activation and the owner of the rule. One issue that arises that is exclusive to heterogenous configurations is *who to notify*, or precisely, which client(s) that should receive the notification of the rule activation. In the default scenario, the user that added the rule should receive the notification. However, the issue of composition can be fully embraced to group sets of clients that share some commonality or goal, similar to engineering notifications in event-based systems [14]. The aim of such composition is to specify boundaries for notification delivery: it semantically restricts the distribution of notifications of a rule activation from the RBS. The boundaries should be able to be specified using a clearly-defined notification semantics.

5 Scoping in Heterogeneous RBS

The solution presented in the Serena framework [1] introduces scoping in heterogeneous RBSes by embracing the concepts of physical or logical groups of clients and their relationships.

Serena models groups internally with the aim of using these representations to enforce data discrimination in the rule engine. It describes a structural representation that uses the notion of a group as a primitive. Serena represents the *group hierarchy* as a directed acyclic graph with the groups as the nodes with the clients connected to different groups at different levels in the graph. It also uses *scopes* to represent the common relationships between groups as a *scope hierarchy*.

5.1 Supported Scopes in Serena

Serena supports the following scope operations, depicted in Fig. 5 with reference to the university security access control example. Constraints imposed by protocols in heterogeneous setups (such as the *classtime access* rule) can be defined using these scopes.

- **subgroupof** (Fig. 5c): Only data added by the specified group or any of its subgroups are included in this scope. This scope is suitable for a departmental rule for *computer science* that will only apply to members of that department or sub-departments (*web info systems, software engineering, bioinformatics*). Its dual is **supergroupof**.
- **visibleto** (Fig. 5a): This scope captures data from clients in groups that have the same ancestor in the hierarchy, e.g., capturing the data that pertains to *senior* academic researchers in one project collaborating with *other* personnel within the same university.
- **peerof** (Fig. 5b): Data items that originate from *peers*, or groups at the same level in the hierarchy, will be considered in this scope. A researcher would for example create a rule with this scope that applies to members in *computer science* and *biology* departments.
- **private** (Fig. 5d): The private scope will exclusively source data from the specified group and none else – not even its subgroups or parent group. This scope is suitable for data that applies to a specific group, e.g., when targeting devices at the campus entrance gates and not those in its related subgroups elsewhere on campus premises.
- **public** (Fig. 5e): The public scope captures all data from all defined groups in the hierarchy. This could be useful for collaboration in the universities by sharing security information between them for data from the devices/student/staff in all the groups.

5.2 Defining Scoped Rules in Serena

Instead of embedding logic for distinguishing clients in the main logic of the rule, Serena exposes scoped rule definitions by extending normal rule syntax with scope-based definitions. The scope definitions specify scope-based constraints on client groups and the relationships between them. A similar approach is observed when enforcing temporal logic in rules: Allen in [16] proposed rule extensions for temporal constraints for point-based or interval semantics, which have been implemented in various systems today, e.g., [17]. The example of protocol 3’s *biologyweekendaccess* rule using scope constraints is shown in Listing 1.3.

Listing 1.3. Scoped rule for biology dept. weekend lab access.

```

1 {rulename: "biology_weekend_access",
2   conditions:[
3     {$stu: {type:"student", name: "?stuname"}},
4     {$stf: {type:"staff", name: "?stfname"}},
5     {$d: {type:"accessdevice", name: "?dev", location:"labs"}},
6     {type:"accessreq", id: "?reqid1", person: "?stuname", time: "?t1", device:
7       ↪ "?dev"},
8     {type:"accessreq", id: "?reqid2", person: "?stfname", time: "?t2", device:
9       ↪ "?dev"},
10    {$test: {$expr: "(hourBetween(?t, 8, 20) && (isWeekend(?t1, ?t2) == true) &&
11      ↪ isNear(?t1, ?t2) )"}
12  ],
13  scopes: [ "biology supergroupof ($stu & $stf & $d)", "$stf private senior"],
14  actions: [
15    {assert: {type: "accessrep", reqid:"?reqid1", allowed: true}}
16  ],
17  notify: [ "subgroupof administrative"]
18 }

```

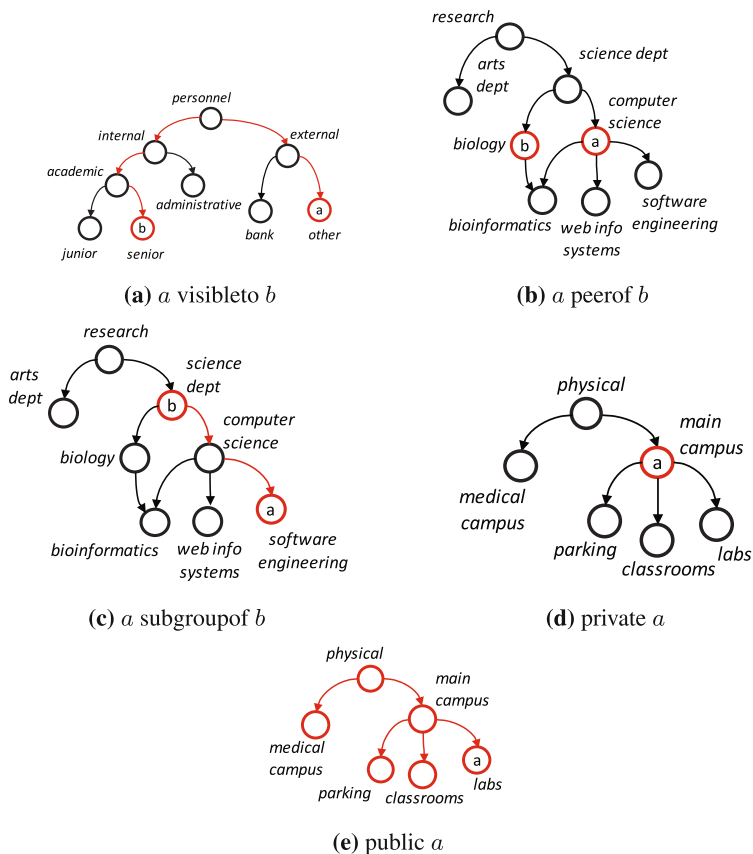


Fig. 5. Scopes supported in Serena [1].

The rule is similar to Listing 1.2. It however has an additional *scopes* section (line 10) where the bound condition variables in line 3, 4, and 5 in the scope constraint are referenced to check whether the student, staff and device facts are all tagged to belong to the general `biology` department using `supergroupof`. The additional scope check in line 10 enforces the constraint that the staff member be from the `senior` academic group. The rule therefore fulfills constraints of Protocol 3 that specified that lab accesses made in the weekends by a student are allowed only if they are accompanied by a senior academic staff member in the biology department, or any of its subdepartments.

5.3 Encoding the Group Hierarchy

Community knowledge requires means in which to determine the compatibility of different sources of heterogeneous data. Rather than performing computationally expensive scope checks (such as path traversals in a hierarchical setting)

Serena builds an encoding that aims to perform near constant-time operations to entirely determine compatibility of data in a heterogeneous setup. This is vital because during the match-execute cycle, Rete can perform combinatorial processing in its computations in the beta network as the dataset increases: therefore client group path traversals will dramatically affect the performance per cycle. The basic idea is that we precompute the scope check, store and maintain them efficiently as an encoding that will be used to expeditiously process scope constraints.

The encoding is based on the *transitive closure*, a significant component modelling most relationships in knowledge and representation systems as identified in [18] that makes our encoding suitable for querying binary relationships – precisely the kinds of operations that the inference engine performs when performing a scope check between left and right inputs. We next outline the encoding process.

The Groups Hierarchy as a Poset. Initially, Serena captures the group hierarchy from an administrator as a partially-ordered set (*poset*). The example hierarchy in can be represented as a poset (P, \leq) with the binary relation \leq defined as ‘*is subgroup of*’, that suffices for most cases.

The poset P has an element (a, b) iff a is part of b . With P we can perform well-defined operations such as calculating the bounds (LUB, GLB) and extrema (maximals, minimals). For instance the maximal in the group hierarchy of Fig. 2 can be represented as a poset (P, \leq) with the binary relation \leq defined as ‘*is a part of*’ (the general \leq relation ‘*is subgroup of*’ is enough for most cases). The poset P has an element (a, b) iff a is part of b , so elements include (*internal, personnel*) and (*computer science, science dept*). The poset however has a limitation of manually searching and traversing the pairs when processing scope operations.

The Groups as a Lattice. A lattice offers improvements over the poset by representing the group hierarchy in a form that is more efficient to encode and compute than the earlier poset representation. The Serena^s framework therefore converts the groups poset to a lattice L , see Appendix A.2. This leads to the hierarchy depicted as the hasse diagram in Fig. 6. Other distinct hierarchies can have their own top-level element same as \top .

Encoding the Lattice. With the lattice L , Serena performs a customised *bit-vector encoding* process that lays its basis on the method by Ait-Kaci [19]. The description is outlined in detail in Appendix C. The process performs calculations for all groups G in the group hierarchy. The result is a *binary matrix encoding* M_{ϑ} of the group hierarchy shown in Fig. 7, with the following properties:

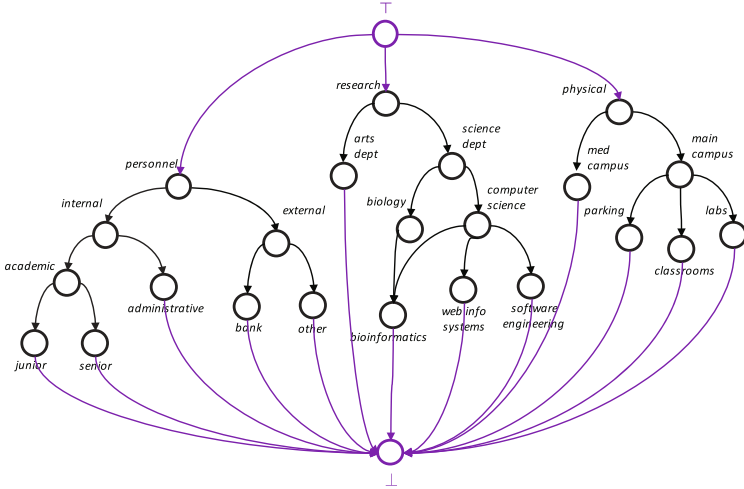


Fig. 6. The lattice Hasse diagram – Serena uses the lattice to encode the hierarchy (Image sourced from [1]).

- (i) The labels on the rows of M_ϑ represent the groups in L ; and similarly for the columns. The first row represents \top and the last row represents \perp .
- (ii) An entry $M_\vartheta(a,b)$ has a 1 if group $a =$ group b or if group b is an ancestor of group a in L , and 0 otherwise.
- (iii) An entry $M_\vartheta(b,a)$ has a 1 if group $a =$ group b or if group b is a descendant of group a , and 0 otherwise.
- (iv) An element a is a *maximal* iff the row $M_\vartheta(a,*)$ has a 1 only at $M_\vartheta(a,a)$ and at $M_\vartheta(a,\top)$.
- (v) An element a is a *minimal* iff the column $M_\vartheta(*,a)$ has a 1 only at $M_\vartheta(a,a)$ and at $M_\vartheta(\perp,a)$.

The process further generates and stores the *level* or depth of each group in the hierarchy. The indexes of all the maximals can also be stored for faster reference.

Scoping with M_ϑ – The M_ϑ is used as the basis of performing scope operations in the rule engine. To facilitate this Serena adds scope tests or guards at appropriate nodes when building the Rete network. The guards are used to perform scoping operations in the beta nodes during the matching process.

- **visibleto:** A scope check of a visibleto b the involves checking if the result of $M_\vartheta(a,*) \wedge M_\vartheta(b,*)$ is a maximal in M_ϑ as per property (iv).
- **peerof:** A scope check of a peerof b includes calculating if $Level(a) = Level(b)$ from the encoding process of M_ϑ .
- **subgroupof:** A scope check of a subgroupof b is true if the result of $M_\vartheta(a,*) \wedge M_\vartheta(b,*) = M_\vartheta(b,*)$ as per property (ii). Conversely, b is a supergroupof a .

	τ	per	res	phy	int	sci	mai	aca	com	biol	adm	lab	clas	sen	soft	bioi	⊥
τ	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
per	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
res	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
phy	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
int	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
sci	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
mai	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
aca	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
com	1	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0
biol	1	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0
adm	1	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
lab	1	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0
clas	1	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0
sen	1	1	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0
soft	1	0	1	0	0	1	0	0	1	0	0	0	0	0	1	0	0
bioi	1	0	1	0	0	1	0	0	1	1	0	0	0	0	0	0	1
⊥	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Fig. 7. The hierarchy as a matrix encoding M_{ϑ} – The rows and columns represent groups in the hierarchy [1].

- **private:** To find out a private b it can check if $M_{\vartheta(a,*)} \wedge M_{\vartheta(b,*)} = M_{\vartheta(a,*)}$ as per property (ii) and (iii).
- **public:** A scope check of a public b includes checking if we $M_{\vartheta(a,*)} \wedge M_{\vartheta(\tau,*)} = M_{\vartheta(\tau,*)}$ from properties (ii) and (i).

During execution, Serena performs scope operations efficiently using these operations. It retrieves the values in the matrix and performs binary operations from the encoding in near-constant time.

5.4 Scoped Execution and Notifications

The matching process stage is where any rule engine performs most of its computation. As mentioned in the previous section, matching in scoped rule engines involves updating the beta network with scope guards that check compatibility of left and right inputs. One way of building the rule in Listing 1.3 is shown in the Rete graph of Fig. 8. The main difference is in the beta node 3 where we now have in place a scope check, a more compact and efficient way to discriminate the tokens for the node to process. The scoping module will use M_{ϑ} to perform the binary operations from the scope guards in the figure denoted with angle brackets.

On a left or right activation, Serena *first* performs the encoded scope check on the fact from the alpha memory or the token’s fact respectively. If the check passes the join computation proceeds as normal. For instance, when a token reaches beta node 3, it triggers a left activation to find a compatible accessdevice. Serena will first perform the supergroup of scope check on the devices as defined in Sect. 5.3. For example, if the access request is made from a device dev in the bioinformatics subgroup, the engine performs the supergroup check on the alpha memory’s device fact, which in this case succeeds:

$$\begin{array}{r}
 M_{\vartheta}(\text{biology},*) \wedge M_{\vartheta}(\text{bioinformatics},*) = M_{\vartheta}(\text{biology},*) \\
 \begin{array}{r}
 10100100010000000 \\
 \& 10100100110000010 \\
 \hline
 10100100010000000 \text{ (biology)}
 \end{array}
 \end{array}$$

Similar operations are performed for the *student supergroup* check and the *private* scope check for the *senior* staff member from the academic personnel group using M_{ϑ} . If successful, we have established the facts are compatible and proceed to the join operation for node 3.

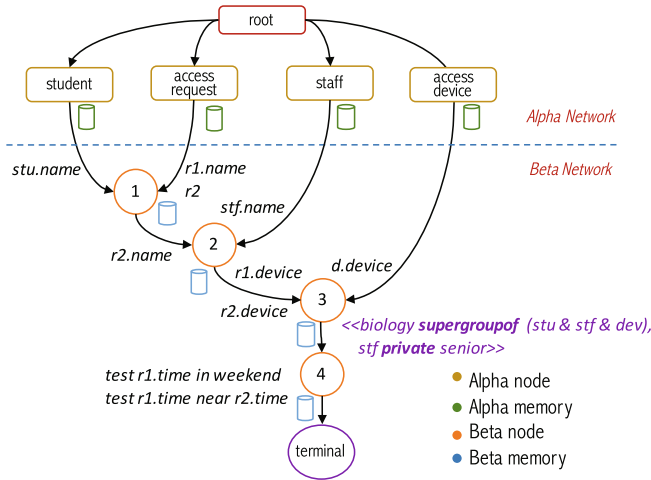


Fig. 8. The Rete graph for *biology weekend access* with scopes – Scopes act as guards that ensure compatibility of data. [1]

The final aspect of rule activation is determining *who to notify*, and specifically, which group of clients should receive a notification. Serena rules expose a `notify` construct that specifies notifications once the rule is fired. In line 14 of Listing 1.3 specifies the groups to notify once the rule is fired. The *notification scopes* are similar to the matching scopes but in this case they can enforce notification constraints to a group, subgroup, or direct clients. Serena invokes similar binary operations as in Sect. 5.3 to determine the groups to notify as when performing a scope check during matching.

6 Experimental Evaluation

The evaluation in [1] used the University Services Access Control scenario detailed in Sect. 1.2 to investigate whether the scoping metadata architecture

has significant computational benefits over traditional techniques in current rule engines. The aggregated results in that work showed evidence of a better overall performance of the scoped engine compared to traditional approaches using expression test and relation facts.

In this paper we compare the scoped approach to the rulebook or *module-based* approach (see Sect. 7) for implementing heterogeneous applications using rule-based systems. We focus on investigating whether scoping has significant computational benefits over the module-based approach in such situations.

6.1 Setup

For the setup, we used the 3 sample universities from the security access control example in Sect. 1.2. We also incorporated similar protocols as prescribed for the scenario. However, as modules represent logically distinct rulebooks, there was need to split the policies according to the relevant university. As expected, in this case some policies were duplicated: for instance, the *classroom access* protocol that applies collectively to all universities needs to be replicated over the modules.

The evaluation was implemented using a simulation running on an event-driven web server. The final application had a total of 61 groups in hierarchies, 40 access rules, and 73 concurrent clients across 3 sample universities. All clients were connected to the server concurrently through websocket connections, with the Node.js server running with an AMD Opteron Processor 6272 at 2.1 Ghz and 20 GB RAM.

6.2 Method

In the simulation, universities receive the same intermittent access requests from various clients with the aim of computing whether the requests deviate from their own protocols as rules. Each simulation was modelled with intermittent requests in ranges of between 1–5 s, limited to 12-h similar to the scope-based simulation in [1]. The requests model students and staff from different departments or personnel levels randomly accessing various university locations. We recorded the resource usage and computations performed during the simulation, and compared the results.

6.3 Results and Discussion

The results of the runs were aggregated and depicted in the graphs shown in Fig. 9. The figure shows the total number of join computations recorded, the number of activations observed and the RSS (Resident Set Size) memory consumed.

The computations in the module instances were observed to be less than those of the unscoped approach – this is because of the additional checks in the unscoped rules that need to enforce data discrimination, and thus lead to

more computations that need to be performed. The scoped approach certainly has fewer computations because it instead uses the encoded matrix to perform scope checks. Remember that join computations have been observed as the most expensive computations in Rete inference engines, Sect. 2.2.

The instances of the module based approach process a higher number of activations than the unscoped approach within the same time interval of 12 h. Indeed, having separate module instances would result in fast computations than the traditional unscoped approach. The limitation of the module-based approach is unearthed by viewing its memory consumption: the memory used by the module instances is significantly higher compared to all other approaches. This is because the instances have increased redundancy, and this leads to the duplication of the working memory, graph nodes and intermediate memories utilised by the rule engine.

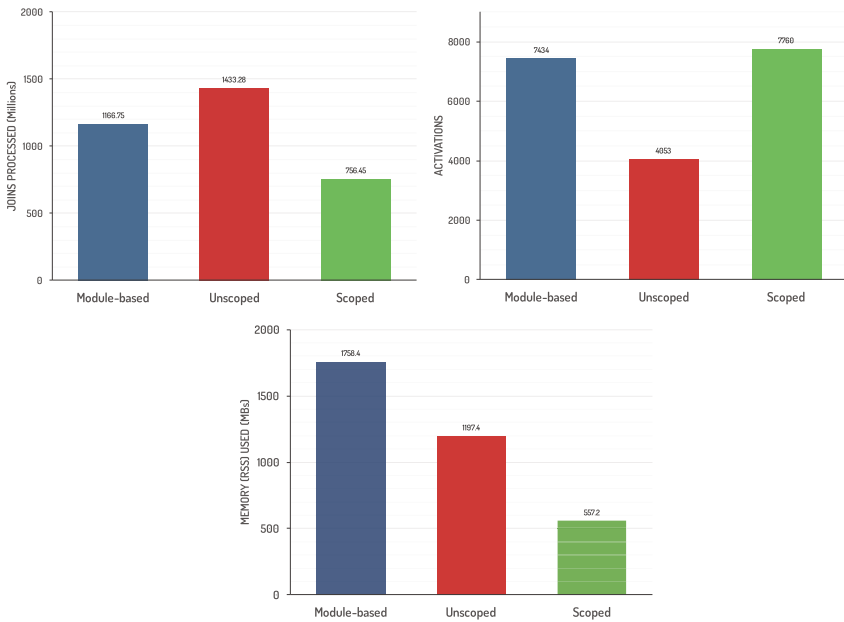


Fig. 9. Results for the experimental evaluation – The scoped rule engine approach performs comparatively similar to the module-based approach and outperforms the unscoped approach. The scoped engine also uses less resources than both approaches.

The results showed that the module-based approach exhibited similar performance as our scoped approach in terms of the rule activations processed: but both outperformed the traditional unscoped approach. The main cost of the module-based approach is however the high amounts of memory utilisation due to duplication of resources. This effectively means that the module-based approach suffers from scalability problems, since more instances would be needed

as clients are added to the system, resulting in significantly higher utilisation of resources. In this sense, our scoped approach enjoys both the benefits of a relatively good performance that utilises less resources during execution.

7 Related Work

We delve into research that provide mechanisms that can be used to control rule engine execution, useful when implementing rule-based solutions in heterogeneous settings. Following the discussions, Table 1 presents a summarised result of the related work in a 5-star ranking system.

Table 1. Comparison of RBSEs and other systems for supporting heterogeneity.

	Metadata Model	Grouping Model	Execution Model	Notification Model
Multitenant Databases				
Shared schema/tables	★ ★ ★ ★ ☆	☆ ☆ ☆ ☆ ☆	★ ★ ★ ★ ☆	☆ ☆ ☆ ☆ ☆
Event-based Systems				
Visibility	☆ ☆ ☆ ☆ ☆	★ ★ ★ ☆ ☆	☆ ☆ ☆ ☆ ☆	★ ★ ★ ★ ☆
Rule-based Systems				
Entry Points	★ ☆ ☆ ☆ ☆	★ ☆ ☆ ☆ ☆	★ ★ ★ ☆ ☆	★ ☆ ☆ ☆ ☆
Peers	★ ☆ ☆ ☆ ☆	★ ☆ ☆ ☆ ☆	★ ★ ☆ ☆ ☆	★ ☆ ☆ ☆ ☆
Rule Modules	★ ★ ☆ ☆ ☆	★ ★ ★ ☆ ☆	★ ★ ☆ ☆ ☆	★ ☆ ☆ ☆ ☆
Scopes	★ ★ ★ ★ ☆	★ ★ ★ ★ ☆	★ ★ ★ ★ ☆	★ ★ ★ ★ ☆

7.1 Rule-Based Systems

Entry Points in JBoss Drools. Drools [2] is a Java-based rule engine based on the Rete algorithm. The engine can deal with event streams that are of high volume and require correlation by receiving and processing inputs from multiple *entry points*. Entry points can be thought of a particular pipe where events from a source flood into the system. New entry points are declared by implicitly referencing them in rule conditions.

When inserting events data into the engine, however, Drools requires entry points to be directly referenced and it therefore lacks a proper meta architecture. Furthermore, entry points do not offer any formalised model for more advanced relationships that can be present in heterogeneous contexts as well as addressing multiple entry-points as a single composable unit. The engine also does not offer any notification model as a feedback mechanism for rule activations.

Peers in Jess. Jess [20] is a Java-based rule engine that can provide reasoning to Web servers as a backend tool. Jess exposes *peering* of its engine, designed to be used in the scenario of pools of Web applications. Conceptually, peers are an evolution of multiple Rete instances. One ‘initial’ Rete engine instance is created and rules added to it, creating the compiled Rete graph. Thereafter, multiple independent *peers* can be created which will share the compiled rules and templates, but each peer contains its own enclosed working memory, execution context and agenda. All peers share the same rule set: changes to the rule set by any of the instances will thus be reflected on the other peers.

Jess’ peering does not provide a metadata model that will effectively manage the different data and sources. The implementation of the separation of instances can be therefore intertwined with application logic, an issue that may complicate the development of heterogeneous applications. Even though peers are instantiated from the initial Rete engine, Jess offers no mechanisms for managing peers that would promote advantages such as addressing multiple peers as a single abstraction. Additionally, when using the peer system in Jess, the programmer needs to ascertain which data will go to which peer. In cases that aim to take advantage of collective intelligence in heterogeneous environments, it is usually unclear to ascertain which peer(s) should receive data from a particular source. Jess also does not expose functionality for managing notifications for responding to clients in the event of a rule activation.

Rule Modules in CLIPS. CLIPS [21] is a Rete-based rule engine that was primarily written in C. It provides modular management for larger rule bases through the use of *rule modules*. A rule module can be thought of as a rulebook, having a set of rules that can be grouped together to leverage explicit control by restricting the access of the enclosed rules by other modules. Modules can therefore be used by rules to control execution. By limiting access to rules, a module functions in the same way as a rule book, allowing facts and rules to be only visible to the module. Each module has its own Rete graph and agenda for its rules.

CLIPS rule modules provide modular abstractions for grouping related rules. However, CLIPS exposes a generic model where facts need to be manually be tagged to a particular rule module during fact data definitions. The CLIPS engine also does not provide any abstractions for managing rule modules that could be used as model for representing the various client structures. Furthermore, constructs are available in rule modules that allow programmers to ‘steal’ the rule engine’s focus to execute a named module. This unnecessarily complicates the programming of heterogeneous rule-based applications, because rule designers are required to orchestrate rule interactions. This unnecessarily detracts the inference engine from its own control of execution. Finally, in CLIPS notifications to individual clients need to be programmed manually.

7.2 Visibility in Event-Based Systems

Event-based systems are increasingly applied in heterogeneous context. Perhaps the closest approach is the work about event notifications in the system REBECA [22]. The designers aim to provide abstractions for structuring event-based systems. The work proposes a way to solve the dialect problem by limiting the *visibility* of notifications in bundled consumers using broker overlays. Only the components that are intended to receive notifications (the intended consumers) are able to ‘see’ notifications filtered by their local event broker.

The work however, only focuses on notification semantics and does not suitably address the actual matching or processing of events. Furthermore, the broker architecture presupposes the existence of some form of an overlay network which requires a more complex management scheme and is not the primary focus of the platform that this work aims to support.

7.3 Schema Sharing in Multi-tenant Databases

Multi-tenant database systems can support heterogeneity by mapping the context of clients into the existing patterns of conventional databases. The closest work in such heterogeneous databases is the shared schema/tables approach. Using this approach the schema is create once and different tenants are mapped directly onto it. This method has the lowest cost and can host the largest number of tenants per server, but has a much higher complexity to implement.

More recent advances have proposed the use of extension tables that reify the concept of a tenant to the database layer, where the database engine can associate each request to a tenant and forwards it to the appropriate storage site. These database schemes are fundamentally designed to process static data and they have static configurations that can degrade in performance when ported to reactive systems with eager incremental processing as with forward-chaining RBSes.

8 Conclusions and Future Work

Community knowledge is significant in today’s heterogeneous systems because of the vastness and diversity of real-time data produced by different clients. This work described a framework that supports scope-based reasoning in heterogeneous systems with the aim of supporting reasoning using community knowledge by mitigating the consistency problems that such systems exhibit.

Scoped rules contain an extended rule-based syntax that allows rule designers to define scope constraints. The work presented uses groups and common relationships between them to build an internal representation that captures the scopes present in many domains. Scopes are a control structure for heterogeneous rule-based languages because they specify a selection of which rules a scope-aware inferencer will consider at a particular time during execution. The scoped model is therefore useful to capture the inherent organisation of client

representations and to define such constraints in rules for the reasoner, thereby harnessing community knowledge in a shared heterogeneous instance. The evaluation showed that the proposed scoped approach enjoys both the benefits of a relatively good performance that utilises less resources during execution.

As future work we would like to investigate support for client-defined dynamic scopes, which will have an impact on the encoding method and the intermediate memories in the Rete graph at runtime.

Acknowledgements. Thierry Renaux is supported by a doctoral scholarship from the Agency for Innovation by Science and Technology in Flanders (IWT), Belgium.

Appendix

A Definitions

A.1 Posets

A poset (P, \leq) is a set P and a binary relation \leq , such that for all $a, b, c \in P$, the following properties always hold:

1. $a \leq a$ (reflexivity)
2. $a \leq b$ and $b \leq c$ implies $a \leq c$ (transitivity)
3. $a \leq b$ and $b \leq a$ implies $a = b$ (antisymmetry)

Poset Operations. Bounds. Given $A \subseteq P$, an element $b \in P$ is called an *upper bound* of A if $a \leq b$ for all $a \in A$. b is a *least upper bound* or LUB if $b \leq a$ whenever a is an upper bound of A . The dual of the least upper bound is known as the *greatest lower bound* or GLB².

Extrema. The *maximal* of a poset P , abbreviated $\lceil P \rceil$, is an element $m \in P$ that is not greater than any other element in P according to \leq . More formally,

$$\lceil P \rceil = \forall b \in P, b \leq m \tag{1}$$

If there is one unique maximal element in P , we call it the *maximum*. The dual of the maximal is known as the *minimal*, $\lfloor P \rfloor$ and a unique minimal is known as the *minimum*.

A.2 Lattices

If in a poset P every pair has at least an LUB \wedge and a GLB \vee , then the poset P with the features (P, \leq, \wedge, \vee) is said to be a *lattice* L . One way to transform the poset P into a lattice is by adding a parent \top to every maximal and a child \perp to every minimal in P .

² The LUB \vee of P is also known as the *join* or *suprema* of A . The GLB \wedge is the *meet* or *infima* of A .

The Covering Relation. We say for two elements $a, b \in P$, a is *covered* by b if b immediately follows a in the poset ordering (i.e. a is an immediate successor of b). More formally,

$$a \prec b \text{ iff } a \leq b \text{ and } \nexists c \text{ s.t. } a \leq c \leq b, c \neq a, c \neq b \quad (2)$$

This enables us to depict a lattice in a *hasse diagram*, where a curve goes from b to a iff $a \prec b$.

Lattice Levels. In this paper we define the level of an element a in a lattice as the longest distance of a from the maximum of the lattice (in this case, \top) to the element, i.e.,

$$Lvl(a) = \begin{cases} 0 & \text{when } a \text{ has no predecessors in } P \text{ and,} \\ \max(\{Lvl(b) \mid b \succ a\}) + 1 & \text{otherwise.} \end{cases} \quad (3)$$

where \succ is the dual of \prec .

B Operations with ϑ

Having L we can define a mapping ϑ from L to another lattice ($S \subseteq, \cap, \cup$) such that for every $a, b \in L$,

$$\vartheta(a \wedge b) = \vartheta(a) \cap \vartheta(b), \quad (4)$$

$$\vartheta(a \vee b) = \vartheta(a) \cup \vartheta(b). \quad (5)$$

If ϑ is invertible, then this makes it easy to calculate \vee and \wedge . $\forall a, b \in L$,

$$a \wedge b = \vartheta^{-1}(\vartheta(a) \cap \vartheta(b)), \quad (6)$$

$$a \vee b = \vartheta^{-1}(\vartheta(a) \cup \vartheta(b)). \quad (7)$$

C Matrix Encoding

We use the encoding method mentioned in [19] taking ϑ as the transitive closure, with a modification that will enable us to map a lattice L to an encoded matrix M_ϑ .

- Instead of starting with \perp , start with \top as the first element. Assign $\vartheta(\top) = 0$.
- Move to the next elements level by level downwards in L and calculate the bitcode of each element as a vector.

- The bitcode of an element $a \in L$ is obtained by

$$\vartheta(a) = 2^{i-1} \vee \bigvee_{a \prec x} \vartheta(x) \quad (8)$$

where i is the number of elements visited since \top , and $a \prec x$ represents predecessors of a ; therefore $\vartheta(x)$ is the code of each predecessor of a .

- An entry in the new matrix M_ϑ for a is the reverse of the bitcode obtained by (8), without the most significant bit.

With this encoding, we can perform operations in Eqs. (6) and (7) having \cap as the bitwise AND and \cup as bitwise OR in M_ϑ .

References

1. Kambona, K., Renaux, T., De Meuter, W.: Reentrancy and scoping for multi-tenant rule engines. In: Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST), ScitePress (2017, to appear)
2. Browne, P.: JBoss Drools Business Rules. Packt Publishing Ltd., Birmingham (2009)
3. Lenat, D., Feigenbaum, E.A.: On the thresholds of knowledge. In: Proceedings of IJCA, pp. 1173–1182 (1987)
4. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al.: A view of cloud computing. *Commun. ACM* **53**, 50–58 (2010)
5. Jamison, S.W., Brereton, J.P.: The Rigveda: The Earliest Religious Poetry of India, vol. 1. Oxford University Press, New York (2014)
6. The Baseline Company: The blind men and the elephant. (http://www.theblindelephant.com/the_blind_elephant_fable.html). Accessed 04 Nov 2016
7. Allee, V.: The Knowledge Evolution: Expanding Organizational Intelligence. Routledge, New York (1997)
8. Fernandez, E., Yoshioka, N., Washizaki, H.: Cloud access security broker (CASB): a pattern for secure access to cloud services. In: 4th Asian Conference on Pattern Languages of Programs, Asian PLoP 2015, Tokyo, Japan (2015)
9. Newell, A.: Production systems: models of control structures. Technical report, DTIC Document (1973)
10. Giurca, A., Pascalau, E.: JSON rules. In: Proceedings of the of 4th Workshop on Knowledge Engineering and Software Engineering, KESE, vol. 425, pp. 7–18 (2008)
11. Forgy, C.L.: On the efficient implementation of production systems. Ph.D. thesis, Carnegie-Mellon University (1979)
12. Miranker, D.P.: TREAT: A New and Efficient Match Algorithm for AI Production System. Morgan Kaufmann, San Francisco (2014)
13. Wloka, J., Sridharan, M., Tip, F.: Refactoring for reentrancy. 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. 173–182. ACM (2009)
14. Mühl, G., Fiege, L., Pietzuch, P.: Distributed Event-Based Systems. Springer, Heidelberg (2006). <https://doi.org/10.1007/3-540-32653-7>

15. Grund, M., Schapranow, M., Krueger, J., Schaffner, J., Bog, A.: Shared table access pattern analysis for multi-tenant applications. In: IEEE Symposium on Advanced Management of Information for Globalized Enterprises, AMIGE 2008, pp. 1–5 (2008)
16. Allen, J.F.: Maintaining knowledge about temporal intervals. *Commun. ACM* **26**, 832–843 (1983)
17. Wang, F., Liu, P.: Temporal management of RFID data. In: Proceedings of the 31st International Conference on Very Large Data Bases, VLDB Endowment, pp. 1128–1139 (2005)
18. Agrawal, R., Borgida, A., Jagadish, H.V.: Efficient management of transitive relationships in large data and knowledge bases. *SIGMOD Rec.* **18**, 253–262 (1989)
19. Aït-Kaci, H., Boyer, R., Lincoln, P., Nasr, R.: Efficient implementation of lattice operations. *ACM Trans. Program. Lang. Syst.* **11**, 115–146 (1989)
20. Friedman-Hill, E.: *JESS in Action*, vol. 46. Manning, Greenwich (2003)
21. Riley, G.: *Clips: an expert system building tool* (1991)
22. Fiege, L., Mezini, M., Mühl, G., Buchmann, A.P.: Engineering event-based systems with scopes. In: Magnusson, B. (ed.) *ECOOP 2002. LNCS*, vol. 2374, pp. 309–333. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-47993-7_14



Bringing Scientific Blogs to Digital Libraries: An Integration Process Workflow

Fidan Limani^(✉), Atif Latif, and Klaus Tochtermann

ZBW – Leibniz Information Center for Economics, Kiel, Germany
{f.limani,a.latif,k.tochtermann}@zbw.eu

Abstract. Scientific blogging is continuously gaining importance in research communities as a complementary support to scientists during different phases of research and publication lifecycle. By enabling early feedback from the community (peers commenting on one’s work as early as the first draft, for example), and providing faster time-to-publish cycles, or tracking audience reach to some extent (via Web 2.0 features, such as shares, likes, etc.), it is becoming an important medium for research(ers). While blogs certainly bring their differences when compared to the more traditional research papers (shorter than typical research articles; include tags – a way of community-driven controlled vocabulary – to aid during blog post retrieval, etc.), they are becoming attractive as complementary scientific resources for Digital Libraries (DL). In this work, we present a complete workflow that spans retrieval, processing, vocabulary handling, and finally integration of a scientific blog posts collection to a DL collection. Moreover, the Use case scenarios that demonstrate the value of the workflow outcome to Digital Libraries along with future development plans are also presented in this paper.

Keywords: Scientific blogs · Digital Libraries · Integration workflow
Linked Data

1 Introduction: Social Scientific What?

Web 2.0, or the “read-write” Web, has enabled higher and easier levels of engagements with the audiences. This, in turn, has stirred individual and group publication initiatives that find these tools practical in terms of networking, collaboration, ease of publication and (unofficial) reviewing from the community, etc. In the case of scientific blogging, for example, authors can easily share their research work with the community, even continuously – as the research develops, be it a result or roadblock that is being shared; whereas the readers are able to provide feedback to this work as it progresses along those results or roadblocks. Burgelman et al. [2] report on “Science 2.0” development, as enabled by tools and changing research behavior practices (spurred by factors

such as lowering entrance barriers, offering processing data and capabilities to wider audiences, etc.), characterized by increased number of authors, publications, and data available to consume, reuse, and comment by the community. In another study, (Mahrt and Puschmann [9]) find “a dual role of blogs as channels of internal scholarly communication as well as public debate” among the motivations for scientific blogging. Furthermore, the same study finds that science bloggers especially value the community feedback on their posts – an additional argument for the development and acceptance of “Science 2.0” from the research community.

Traditional publication repositories have already moved on to embrace the benefits of Semantic Web technologies. Projects that structure and represent Digital Library (DL) repositories as machine-readable, and link them up and make them available to the Linked Open Data (LOD) cloud are pretty common (according to the “State of the LOD Cloud 2017”¹, publications take the second largest data set on the LOD). The Library of Congress Linked Data Service² offering standards and vocabularies used by the library; the British Library’s LOD initiative³; the Swedish National Library Open Data project including bibliography and authority data⁴; German National Library of Economics’s EconStor LOD project⁵; are just some of the LOD projects from the domain of DL repositories.

As scientific blogging is getting more contributions and prominence in the research community, we see major benefits from putting all the author, publications, and research data contributions to use in different scenarios and environments. In this work we focus on (1) Integrating them with the more traditional DL publication archives, and (2) “Porting” scientific blogs on the Web of Data for supporting applications benefiting from these resources in the future.

2 Motivation and Use Cases

2.1 Motivation

After many requests from the scientific blogging community, the German National Library of Economics (ZBW) is considering the opportunity of extending its repository by including scientific blog posts from the domain of economics and offer it to its users alongside the standard research publications. This is the single most important motivation for this study. This motivation is just part of a broader picture of increasing scientific blog contributions and their adoption in the scientific work flows, which furthermore emphasizes the importance of publications from the scientific blogging community.

¹ <http://lod-cloud.net/>.

² <http://id.loc.gov>.

³ <http://bnb.data.bl.uk>.

⁴ <http://libris.kb.se>.

⁵ <http://linkeddata.econstor.eu/beta>.

2.2 Use Cases

Following are the main use cases that motivated our research

1. *Heterogeneous Data Integration*: Blog collections do not adhere to a standardized metadata structure and often rely on different vocabularies from the ones adopted by DLs. In a situation like this, a user interested in resources in both DL and scientific blog resources would have to query these collections separately, using the different vocabulary terms. Thus, there is an opportunity to alleviate this situation and combine these collections in a uniform “query space”. EconStor, our DL of choice, metadata are already structured and represented in RDF [6]. As a framework, this representation is well equipped to handle combination of heterogeneous resources.
2. *Semantic Annotation of Blog Post Collections*: More and more resources are being added to the Web of Data, benefiting in this way both publishers (making their resource machine-understandable and available to an additional pool of users) and consumers of those resources (for example when integrating these resources in environments of interest, DL repositories in our case). In case scientific blog post publishers are interested in making their content available on this platform, they should not be concerned with any technological barriers to achieve this goal.
3. *Dataset Profiling*: If “isolated”, the scientific blog post collection has limited value for the user. Linking these resources with relevant entries in external collections and knowledge bases (KB), indexed using different controlled vocabularies (CV) – thesauri and classification schemes, in our case – from the one used by scientific blogs, but that have been aligned with, is another value-adding step for the end user. In this way we provide different “profiles” for every blog post in our collection, all depending on the external resource(s) it links to, enabling the user a more elaborate and rich (search) experience. The user search experience could see improvements from related resources to their current reading from the fields of economics, social sciences, or agriculture; retrieve additional information (and context) from a KB, such as DBpedia; include relevant resources from a German-specific or international, multilingual collection; etc.
4. *Dataset Analysis*: Summarizing datasets by offering useful statistics as exploration tips for users is quite important. This especially holds for large datasets that could prove challenging for users to grasp in order to use them to their full capacity (e.g., identifying resources of certain features, closer to their area of interest). In this regards, scientific blog posts semantification lends us various analysis options, such as highly commented/discussed (expressed via user comments to) blog posts; most featured/covered subject in the collection (based on their topic coverage); “trending” subjects for a given period of time (based on the number of blog posts for a given subject); the top contributing authors per subject/topic, which would, in a way, identify expert groups on certain subjects/topics; or, in the context of aligned CVs of different (linked) datasets, suggest publications by an author in external KBs; etc.

3 Related Work

Blog post collections require established publication and dissemination infrastructure, as well as increased visibility of their content. DL repositories have this, and would consider an added value by offering them alongside their original bibliographic content. There is solid research done on mapping relational data (RDB) to graph representation, as well as publishing and integrating heterogeneous collections, part of which also includes social web data. Furthermore, DL repositories are more and more moving to the Web of Data making this possible inclusion even easier with already established approaches and practices.

Auer et al. [1] present common motivations for representing RDB in Resource Description Framework (RDF) data model; the use cases for integrating RDB with structured sources or existing RDF on the web (Linked Data⁶) correspond to a great extent with our motivation for this work. Moreover, Spanos et al. [13] survey the proposed approaches for mapping and integrating RDB content to/with the Web of Data, with citing semantic annotation of dynamic web pages and mass generation of Semantic Web data as some of the key motivations of the work.

Holgersen et al. [3], with their semantic-web based framework, part of the Swedish “The Open Library” project implementation, transform social content repository data (user-generated content in the form of tags, reviews, comments, etc.) in a Resource Description Framework (RDF) machine-readable format, to further enrich it by related data from external collections, and provide (as a Web Service) the resulting data to libraries that hold related bibliographic records to benefit from (this enrichment). As a result, libraries can associate and show all their bibliographic records and related social content to their users. Hu et al. [4] publish a journal collection based on Linked Data principles, which, besides bibliographic data, includes information on the publication submission and review process of the journal, such as reviewers’ comments, editors’ decisions, author replies, to name few of the (meta)data. The data is further enhanced by relevant resources from the LOD Cloud, such as DBpedia and Semantic Web Dog Food⁷, and made available as part of the LOD Cloud⁸.

Atif et al. [6] go through both conceptual and practical aspects of publishing an Open Access (OA) repository as Linked Data. To demonstrate the potential from this undertaking, they link up the repository collection with the LOD Cloud datasets (via mappings to an economics thesaurus, lexicon-related service, and an economics classification system – all part of the LOD Cloud) for contextualizing the OA repository collection and enabling more discerning queries to be conducted over it.

Powell et al. [10] demonstrate fusing library and non-library data from disparate resources by applying Semantic Web technologies to the task. They rely on RDF as a common data model, and use graph-based analysis and visualization

⁶ <http://linkeddata.org/>.

⁷ <http://data.semanticweb.org/>.

⁸ <http://lod-cloud.net/>.

to generate useful information on the resulting data for the user. In another work, Yoose and Perkins [14], in a survey on LOD adoption in libraries, including archives and museums, report important and increasing number of L(O)D projects that free library resources from specific library representation formats, benefiting the consumption of this data from interested parties, and further improve the library experience for its users by enriching this (meta)data with relevant resources from the LOD Cloud datasets.

Passant et al. [11] research the application of Semantic Web technologies in an industry context for enhancing the Enterprise 2.0 setting. In their study they present a paradigm for reusing collaboratively-built knowledge in the enterprise, contained in different fragmented information resources and represented across heterogeneous data formats, which, to also provide more insightful query capabilities to the end users.

Due to the said prominence of blog posts, there are other research efforts for collecting and developing value-adding services based on these resources. In a related undertaking, Papadokostaki et al. [12] develop a platform for storing, indexing and searching blog posts. In their effort they apply semantic web technologies throughout the project, starting from developing a custom ontology for modeling the blog posts, storing the resulting posts (as RDF) in a triplestore, as well as relying on a Linked Data format (JSON-LD) for their REST services; when modeling the domain - news articles, the authors develop a new ontology. All blog posts added to the collection are also made available to the Linked Open Data cloud. Users of the portal can benefit from searching the blog post collection created by automatic ingestion of blog posts, or have the possibility of manually adding blog posts to the collection. The BlogForever platform (see Kalb et al. [5]) focuses on archiving operations for the blogosphere; in addressing the potential “information silos” from this undertaking (blogs from different domains, using different vocabularies, etc.), the authors adopt Linked Data principles in their approach. A common domain model for blog archiving and a set of vocabularies enable exposing resulting blog archives as LOD. The benefits from this work are manifold, such as: possibility to search across different blog archives; link archives to external resources or collections; etc.

Harnessing the value from connecting heterogeneous resources – in this case blogs and DLs – is specifically the motive for our work in this paper. In this work, we integrate blog post collections external to and independent of the DL collection for increased visibility to the former, and enriched offer of the latter. Blog posts typically are not pre-related or do not refer to DL publications (such as via a publication DOI, or other identification mechanisms); and bloggers write on any topic they deem important, regardless of the DL repository publications.

4 Methodology

This section details the applied methodology, including the dataset selection for this work, its (pre)processing and augmentation, modeling and conversion, as

well as enrichment (via linking) from other resources. For additional information pertaining to Sects. 4.1 and 4.2, we refer you to our previous work in Limani et al. [7]).

4.1 Data Selection

In order to support our use cases, our dataset selection contains a DL repository and a blog post collection. A final component, a thesaurus, is also presented here for contextual information; its role will be detailed as we go further in the section.

1. *DL Repository*: Our requirements for this part are open access policy of the repository holding the DL collection, and potentially the existence of a controlled vocabulary in order to emulate the DL environment as closely as possible. The former eases the aspects of retrieving and using collection resources, whereas the latter brings up the challenge of streamlining resources with different vocabularies. EconStor⁹, the DL representative choice for this work, is an open access publication platform for the domain of economics and related fields, supporting publication (at the time of writing with more than 140 thousands working papers, journal articles and conference proceedings, etc.) dissemination for many institutions, as well as providing its collection meta-data to other academic repositories.
2. *Scientific Blog Collection*: For this part we needed qualitative and rich blog collection to pair up with the standard that EconStor collection employs. With over 40 K blog posts from the domain of economics that we harvested for this work, we chose the The Wall Street Journal¹⁰ blog for our work. A few notes are in order to provide the rationale of our choice of the WSJ as our scientific blog representative. There are two main reasons for our decision:
 - (a) Although not scientific blog per se, being a popular authority on economics publishing, it provides a useful resource for a DL that covers the economics (and related fields') domain. We feel this could be a real use case that users of the DL will appreciate (and there is an evaluation plan to assess this as our future work); in the same way, other similar publications – relevant and of high quality as the DL collection – could serve in the same way for our research process workflow.
 - (b) For our workflow we needed a rich selection of blog posts in order to demonstrate its usefulness in a DL setting. For that we needed a single, rich blog to save us from having to deal with different approaches especially during the blog posts retrieval, and one with regular updates and high-quality posts and relevant authors. The WSJ is just one of the many such collections that offer blogging on relevant and up-to-date topics from the economics domain.

⁹ <http://econstor.eu>.

¹⁰ blogs.wsj.com.

3. *The Standard Thesaurus for Economics (STW¹¹)*: With about 6.000 descriptors (both in German and English), it is a rich vocabulary primarily covering the domain of economics, as well as related fields (law, sociology, politics, etc.), used for indexing and retrieval operations for EconStor publications. Furthermore, its alignment with other CVs and collections increase its “reach” and value: thesauri for social sciences and agriculture (TheSoz and Agrovoc, respectively); classification system for publications from the fields of economics (JEL), as well as with publications used in German libraries (German National Library Integrated Authority File); or even relations to a KB (DBpedia). These alignments are important to mention as they enable the realization of some of the benefits listed in Sect. 2 of this paper, in “Dataset profiling” and “Dataset analysis” subsections.

4.2 Dataset (pre)Processing and Augmentation

The “Data selection” subsection emphasized on the differences between selected data sets – DL and scientific blog collections. To account for this aspect, our methodology includes activities that (i) bridge the “terminology” gap between the heterogeneous data sets, and (ii) identify blog post metadata that are important for our use cases. Specifically, we conduct:

1. *Automatic Term Assignment based on a CV*: Having to deal with heterogeneous resources brings its own challenges to the table. Namely, after having selected our data set as described in the previous subsection, we need to work with their inherent differences seamlessly. With vocabularies used in either collection being the most prominent difference, we need to be able to conduct all our analysis on these collections as if it were a single, homogeneous data set, regardless of any of the differences mentioned so far. Having the DL data set (EconStor) described by the STW thesaurus determined a similar requirement for the blog post collection – i.e., describe it using the same set of vocabulary terms. Due to the fact that blog posts are created at a frequent pace (every blog could have many posts added every day), we needed to automate this step. In this work we used a mature and popular automatic indexer (MAUI¹²) to automatically assign terms to every WSJ blog post. One advantage that MAUI gives us is the possibility to specify a CV as a source from where to choose terms from during the term assignment process. This is exactly what we needed for our case: MAUI suggested terms to WSJ blog posts using the STW thesaurus as a source. Examples of this term assignment can be found in the “Process workflow” section.
2. *Blog Post Metadata Selection*: Blogs differ in the attributes they employ in their post collection. In our case, besides the usual blog post metadata, such as author, title, content, and publication date, we also retained blog post comments for each blog. Some social web features, such as “shares” and “tweets”,

¹¹ <http://zbw.eu/stw>.

¹² <https://code.google.com/p/maui-indexer/>.

although present in the data set, were left out of the current research analysis. We have plans to include them in our future work.

4.3 Dataset Modeling and RDF Conversion

This part of the methodology covers three activities that complete the data set modeling and conversion process, including: (i) listing selected vocabularies for the blog post collection, (ii) providing the technical details of mapping the blog post collection from a relational database to RDF¹³ via a platform that “wraps” the RDB into a virtual RDF graph; and (iii) combining EconStor and WSJ blog post collections in a single collection, to be used as a unified query space; this activity also provides for query and exploration capabilities via an HTTP server for RDF data.

1. *Blog Post Vocabulary Selection:* Despite the Web 2.0 nature, blog posts reflect the common metadata that a DL publication has, such as post title, publication date, content, terms describing the post; whereas also having some additional aspects that are inherent to them, such as user-generated feedback (blog post comments and other Web 2.0 “features” like shares, tweets, etc.). The key vocabularies selected for modeling blog post collections are SIOC¹⁴ (and SIOC Types module) and the Dublin Core Metadata Initiative¹⁵; the former covering especially well the user-generated content, whereas the latter the typical publications metadata. An example representation of a blog post is shown in Fig. 1 below, with a single comment and subject keyword (“Marketing”, in this case) due to easier readability. One of the key parts of the modeling refer to blog posts and related comments, with SIOC Types `BlogPost` and `Comment` classes used for the blog post and their comments, respectively. The modeling requirements of blog posts did not necessitate developing a new ontology, so we reused already established vocabularies for this process.
2. *Mapping Relational Data to RDF:* The DL collection is represented via Resource Description Framework (RDF) standards as (RDF) triples, whereas the scientific blog post collection is stored in a relational database. To bring both collections into the same model, we use a specific platform to generate an RDF data dump of the blog post collection. This requires mapping the relational database tables and columns to RDF classes and properties, based on the vocabularies identified beforehand. With this both data sets are combined in a single data model – RDF.

Figure 1 displays the key classes and properties used to model the blog post collection. The modeling is based on three “entities” of a WSJ blog post: the blog post itself, its comments, and the subject it covers (represented here based on the STW terms automatically assigned to the post). The blog post

¹³ <https://www.w3.org/RDF/>.

¹⁴ <http://rdfs.org/sioc/spec/>.

¹⁵ <http://dublincore.org/specifications/>.

entity and comment entities are mapped to SIOC Types BlogPost and Comment classes, respectively; the last class - that of blog post terms - uses a D2RQ platform-internal “vocabulary”. The properties used to describe this entity, i.e. the subject it covers and the link to the STW URI, were enough to support our use cases; we did not find it necessary to use a vocabulary to specifically model the blog post terms beyond the one that D2RQ platform provided during mapping.

3. *Unified Query Space over Data Sets:* At this point both collections use the STW thesaurus to describe their resources; this implies a closer connection of the data sets and renders them more integrated than just two data sets sharing the same representation (RDF, in this case). Rather, this enables us to address the combined data sets as being part of a single “information space”. In this way, we can solely rely on the STW terms to search resources in both data sets.

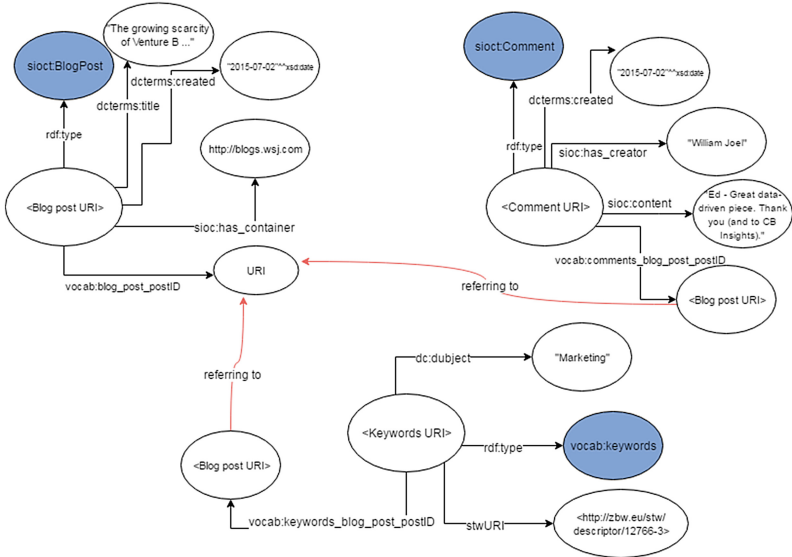


Fig. 1. Classes (colored in blue) and properties modeling a blog post instance. Source: Limani et al. [8]. (Color figure online)

5 Tying It All: A Process Workflow

After discussing the methodology followed in this work, in this section we turn to a concrete implementation - represented as a process workflow which is depicted in Fig. 2. In following, we provide technical details and rationale for every process of this workflow. Concretely, we show the complete process of making scientific blog posts available to the DL environment, and provide technical details and

information about the tools used to achieve this goal. We believe that this could help anyone planning to apply or extend the workflow according to their needs for their DL environment or in a completely new project. We provide some potential extension examples as we go through the process.

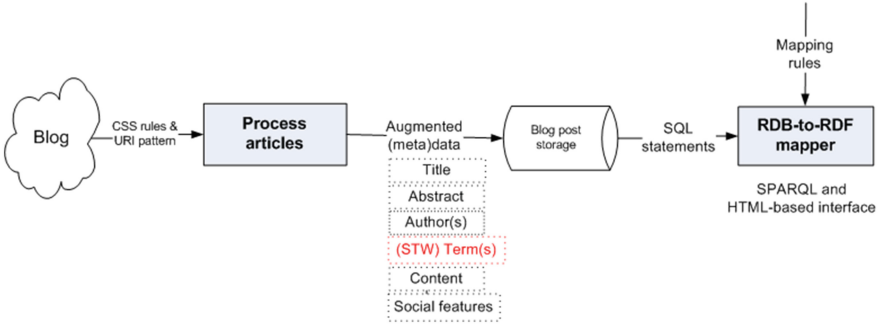


Fig. 2. Process workflow: from “raw” blog posts to access within DL.

1. *Data Set Retrieval:* EconStor is already available as RDF and regularly updated; furthermore, it is also available for download (as RDF data dump). The WSJ blog post collection, on the other hand, needed some effort. Based on the URL pattern of the WSJ blogs, we retrieved all blog posts from the different categories that WSJ supports. Blog post collections are more dynamic in nature, as they get new posts and updates (comments, shares, etc.) to existing posts. We retrieved posts of a certain period, and that suits our experimentation purposes.

Potential extension for this step: Depending on the datasets of interest, the retrieval process could entail different approach, such as using an API that makes resources of interest easily available.

2. *(Pre)Processing Blog Posts:* After retrieving a single blog post, we extracted elements of importance to our experiment (as described earlier in the paper) by relying on the CSS description that WSJ employs for the different post elements. Sometimes the author names are part of the post contents. We relied on NLP operations (Apache OpenNLP¹⁶ in this case) to identify (and remove) author names from post content. A final operation for this process adds to the list of elements by (automatically) assigning STW thesaurus terms to every blog post; MAUI was configured to assign up to 5 terms from the STW to every post, and the result is stored in a RDB.

Just to afford the idea of this last operation, Table 1 lists few examples from the process of automatic term assignment with MAUI. The first column stores the title of the blog post; with it, we can guess to some extent what the topic of the post is. The second column stores the terms (or tags) that are used

¹⁶ <https://opennlp.apache.org/>.

to describe the post; these tags are specific to WSJ blogs, and are used by its authors and readers to find posts of interest. The third column represents the terms that MAUI automatically assigned to every blog by choosing the terms from the STW thesaurus.

Potential extension for this step: Any other pre-processing activity that one could need, such as: using a different (automatic) indexer and/or CV to assign terms to resources; applying different enrichment activity, such as linking a resource to an external KB, for example; using any NLP-based operation that the new domain/application could require; etc.

Table 1. Several blog posts as described by their tags and the STW thesaurus after automatic indexation.

Blog post title	Original terms	Auto-assigned terms from STW
The Growing Scarcity of Series B Venture Rounds	Culture; Investors/Raising capital; Investor	Marketing; Market; Seed; Enterprise
Fueling the Next Generation of Innovation	Business model; Culture; Entrepreneur	Enterprise, Needs, Innovation, Fuel
Understand the Risks of Going Public Before You Ring the Bell	Business model; Culture; Investor	Risk; Listed company; Publication; IPO
Why Design Matters More than Moore	Culture; Customer acquisition; Investor; Weekend read	Designers; Engineering; Technology; Metalloid
Don't Fall in Love with Your First Idea	Business model; Culture; Entrepreneur; Getting to Product Market Fit	Emotion; Marketing; Marketing; Joint production

3. *Access and Query:* The D2RQ platform¹⁷ “wraps” a RDB and provides access to it as if it were a graph DB. In our case, since the blog post collection was stored in a RDB, we decided to use D2RQ. One prerequisite to this option is to provide a mapping file that D2RQ will use to map RDB columns to graph representation. This way, we can have a “direct” access to the blog post collection as if it were a graph, or generate a RDF data dump to be used later on. We choose the latter to have the complete blog post collection as RDF data set, later to be combined with the DL collection (already available as RDF).

Potential extension for this step: The datasets could be directly represented as RDF, thus eliminating the requirement for any intermediary tools for access and query of datasets. For example, having the datasets stored as RDF in a triplestore solution, as opposed to “extending” RDB solution’s feature set to

¹⁷ <http://d2rq.org/>.

provide SPARQL access. Moreover, in case other/new vocabularies are applicable for a different domain or a new application, one can always change the mapping rules in the D2RQ platform to accommodate for this requirement. In this case, as one would expect, changes introduced should be accounted for during the query process.

4. *Availability and Storage of RDF Data Sets*: Now that both data sets are represented in the same way (RDF graphs), and use the same term vocabulary (STW thesaurus), we needed a way to integrate them in a single data set and demonstrate the use cases envisioned for this work. We choose Apache Jena Fuseki SPARQL server¹⁸ for this purpose.

We loaded both data sets as a single data “unit”, with 2 different named graphs in order to provide flexibility when implementing use cases, such as when required to access or query only one of the data sets. Such was the case in one of our use cases where we wanted to select relevant blog posts after the DL user has accessed a publication from the DL collection. In the use cases section you can see more on the usability of the separate graphs for the two data sets.

Potential extension for this step: Depending on the technical choices from the previous activities of the workflow, one can choose a solution for this step, such as graph, NoSQL, or any other storage solution that suits their needs.

5. *Vocabulary Bridging: Any Ontology Alignment/Matching?*: One of the key decisions planned in our methodology and implemented via our process workflow is the one regarding the challenge stemming from having to integrate different vocabularies, DL and blog posts in this case. For example, EconStor uses a thesaurus to describe and base information retrieval services on it, whereas the WSJ uses its own categories and terms to describe posts. Furthermore, even for blogs themselves, there is a difference in terms of vocabularies used. This makes it a real challenge to try to align or map vocabularies used in a DL with all the different vocabularies that could be employed by blogs.

This is where the automatic indexer comes into play: by using the STW thesaurus to automatically assign terms to all blog posts (currently, we are only using WSJ posts, but a DL will most likely include more blogs), we effectively bridge the terminology gap, thus eliminate the need to pursue other alternatives, such as ontology alignment or mapping between the STW thesaurus, in our example, with the vocabularies adopted by the blogs.

Potential extension for this step: One can choose to develop a vocabulary that specifically covers the domain of blogs (as some of the related work presented in the paper have chosen to do), and then try the arsenal of approaches of ontology alignment/matching to bridge the vocabulary gap.

6. *SPARQLing for More Insight*: One benefit from our methodology, especially the dataset representation in RDF, is the capability for more insightful queries. For example, one could be interested to know the female authors active with blog posts during a certain period of time on the topic of

¹⁸ <https://jena.apache.org/documentation/fuseki2/index.html>.

“technology transfer”; or some of the most commented (shared, or liked, etc.) blog posts during the financial crisis of 2008, and similar queries. The fact that it is the query language for Linked Data, opportunities for more perceptive querying are great.

The following SPARQL code listing shows an example that lists WSJ blog posts and EconStor publications related to the term “Technology transfer” published since 2014/2015. The code queries the named graphs – `econstor` and `wsj` – with the same query, and “combines” the results (note the UNION operand) into a single result list; namespace definitions are omitted for brevity.

Potential extension for this step: The query can be refined by specifying narrower or broader terms from the STW thesaurus, in case there are many or few results from the query; filtering results based on a number of parameters, such as comments, tweets, shares, etc.

Listing 1.1. A SPARQL example querying heterogeneous datasets.

```

SELECT ?publ ?title
From Named <http://localhost:3030/EconstorWSJ/data/econstor>
From Named <http://localhost:3030/EconstorWSJ/data/wsj>
WHERE {{
  Graph ?g {
    ?publ a ?o;
    econStorDC:issued ?date;
    econStorDC:title ?title;
    econStorDC:keyword "Technology_transfer" .
    Values ?o {swc:Paper}
    Filter(?date >= "2015"^^xsd:gYear) }}
UNION {
  Graph ?g{
    ?publ a ?o;
    dcterms:created ?date;
    dcterms:title ?title .

    ?term a vocab:keywords;
    dc:subject "Technology_transfer";
    vocab:keywords_blog_post_postID ?publ .

    Values ?o {sioc:BlogPost}
    Filter(?date >= "2014-01-01"^^xsd:date) }}}

```

Our proposed methodology, and the process workflow implementation driven by it, can be adopted/adapted for similar work by other DLs that seek to integrate emerging scientific publications into their collections. In our case we dealt with a DL that relies on a domain-specific thesaurus, and plans to integrate blog posts from that same domain. However, new use cases and possibly completely new applications, thus new extensions to the workflow are more than encouraged to pursue by interested parties.

6 Use Case Scenario Demonstration

In this section we represent several use case scenarios implemented from integrating the DL repository and the scientific blog post collection.

Although DL users are not expected to know SPARQL in order to search the collection, by exploring some query scenarios, we want to demonstrate that this collection can serve as a data store on top of which we can build a standard user interface for search that users understand (keyword-based search, in the same way they use a search engine or search a document in their computer). While SPARQL can support insightful queries of the data, we would like to also mention at this point that the queries we show are constrained by the metadata in both dataset. For example, we could explore the contribution of female bloggers from a certain region, during the “financial crisis” period, for example, had he had the (meta)data in our datasets.

1. *Search Across the “Unified Query Space” (EconStor and WSJ datasets):* The user searches for publications related to the subject of “technology transfer” in EconStor and WSJ datasets. As mentioned earlier, there are several types of publications archived in EconStor, but, in this case, the user is interested in research papers (i.e., `swc:Paper`), published since 2014. The search returns four results in total, with three results coming from the EconStor dataset, and one coming from the blog post collection. This just demonstrates the possibility for the user to search across two different datasets described with the same thesaurus term(s), and see a result of publications from corresponding datasets (treating the datasets as if they are one source of information). The following listing shows the SPARQL query that implements this use case scenario.
2. *Retrieve Relevant Blog Posts for a DL Publication:* This scenario is related to the previous one: the user initially searches the DL collection (i.e., EconStor) and selects a publication that she wants to further examine. We search the blog post collection for additional publications that could be of interest to her based on the STW term(s) that describe the publication she is currently reading. The user searches for (`swc:Paper`) publications from EconStor that cover the subject of “Human capital”, published from 2014 and onward. The user selects the publication titled “Labour market integration, human capital formation, and mobility” from the result list. Using the same STW term (“Human capital”) that describes the selected publication, gives us 1 blog post from the WSJ collection, titled “Q&A: Golub Capital’s David Golub on GE Capital’s Divestiture”, as well as 7 other posts described with the “related” STW term “Human resources” that could further complement user’s reading experience. This further emphasizes the role that the (STW) thesaurus can play in providing alternative results for the user by using its structure, such as via “narrow”, “broad”, or “related” terms.
3. *Search the Scientific Blog Post Collection Alone:* In another scenario, the user searches for the newest blog posts covering a certain subject. During this scenario, the user can decide to factor in the number of comments that a blog post has, i.e. the post that stirred the most feedback/discussion on a given subject, or explore the most used STW terms from the collection, in order to have a understanding on the variety of blog posts that constitute the

collection. Let's see how these two search strategies work for our blog post collection:

- (a) **Highest Number of Comments:** This search, filtered by posts published from 2015 and on, lists the following top 3 blog posts with the highest number of comments: "Facebook Plans a 'Dislike' Button, but Only for Empathy, Zuckerberg Says" with 18 comments; "Microsoft Expected to Unveil Next-Gen Windows Phone and Surface Tablet" with 12 comments; and "Alabama Judges' Reprehensible' Conduct Merits Impeachment, Judiciary Says", the last post stressing a judicial misconduct by a judge, with 8 user comments from the blog readers.
- (b) **The Most Featured Blog Posts by STW Term:** This is an attempt to mimic "topic trending" in the blog post collection – showing the extent to which certain subjects are covered (via posts) in the blogging community. In our case, searching for the most used STW terms in the blog post collection results with the top 3 most used terms "Enterprise", "Personalization", and "Share". This provides some hints to DL users about the most represented/covered subjects from the blog post collection, in case they want to use that information to guide their exploration of this collection.
- (c) **A Combination of the Two:** Having identified the most used STW terms, the user can further explore the most commented on blog post from a popular subject, which with regards to our blog post collection results to combining posts on the subject of "Enterprise", "Personalization", or "Share" (as discussed above), and blog posts that attracted the most attention in the blog community.

7 Benefits

The key benefits relate to the automatic indexing and semantic annotation of the scientific blog post collection, its integration with the DL collection in a unified (in terms of querying and resource description via the STW thesaurus) dataset, as well potential data profiling and analysis operations. Following are the emphasis on these aspects:

1. *Semantic Annotation and Representation of Blog Post Collections:* Having the DL collection published as LOD dictates the methodology of blog post collection integration with the DL. Without any effort from the bloggers' side, we have modeled and represented this collection in the same way as the DL collection – thus making them part of the same "model" (RDF, in this case), and automatically indexed it (based on the STW thesaurus) – thus bridging the terminology gap between these different resources and integrating them at a terminology level.
2. *Integration of Heterogeneous Collections in a Unified "Query Space":* Meeting DL's interest to include heterogeneous resources – blog posts from the same domain, we have integrated the latter and made it available as a resource collection to the former. The users of the DL library, as shown with our queries

over the resulting dataset collection, are able to retrieve relevant resources via different scenarios.

3. *Data Profiling and Analysis*: both indirect benefits from relying on STW for indexing the blog post collection. STW's alignment with other thesauri, classification systems, and external KBs enables us to enrich the user search experience by linking up scientific blog posts of interest to the user with external, related resource collections. Moreover, we are able to provide useful information about the dataset to the user, such as “trending” topics/subject for a given time period, the most popular topic/subject, or the blog posts that sparred the most debate with the users. For more details, see the implemented use case scenario implementations from Sect. 5 of the research paper.

8 Conclusion and Future Work

In this section we present our final thoughts on the research endeavor, as well as present its follow up steps for the near future.

- *Summary of Conclusions*: With our undertaking we have addressed a DL requirement for integrating non-library resources a scientific blog post collection and making it available to its users as a complementary or additional content in their search operations. In doing so, we have pre-processed the non-library resources in order to bring them up to par with vocabulary-wise with the DL practices (assigning STW terms, in this case); modeled them according to the DL collections representation (RDF, in this case), by selecting a set of suitable vocabularies (and corresponding classes and properties); and finally converting them from a relational database to an RDF representation using the D2RQ platform. Furthermore, in order to support the use case scenarios, we loaded both the library and non-library datasets on separate named graphs of a single dataset on a SPARQL server.
- *Future Work*: One of the follow up efforts is developing a prototype enabling evaluation scenarios with the users. Furthermore, we would also like to compare our algorithmic approach (i.e. for generating recommendations of blog posts) with state of the art similarity measures from Graph-based and conventional Machine Learning approaches. We are of view that this comparative evaluation will help us to improve the accuracy and recall of our generated recommendations. Currently, in order to query the unified dataset implies knowledge of SPARQL, which is not a skill that common DL users should have in exploring a DL collection.

Another research follow up direction is that of analysis that would bring more value to the user (search) experience in view of the newly-added blog post collection, such as publications similarity based on the STW thesaurus structure and graph representation properties, to name a few.

There is also work planned regarding the evaluation of our work. Some of the preliminary research questions are to be directed towards establishing the complementarity of blog posts to DL resources, with the underlying hypothesis being that blog posts add value to and provide the serendipity effect for

DL users. The prototype should be able to answer these initial questions, as well as raise new questions for the final evaluation (handling information overload could be such a question, that could potentially hamper or cancel the role of blog posts).

- *Limitations:* During this phase, we are relying on SPARQL to explore the resulting EconStor and WSJ blog post datasets, although a typical DL user does not and should not have to have any knowledge of SPARQL or, at a more general level, Semantic Web technologies to access and use DL services. A solution involving a graphical user interface would enable a more comfortable environment for users, and enable them reap the benefits of our research without the higher technological barrier that Semantic Web technologies represent for common users.

References

1. Auer, S., Feigenbaum, L., Miranker, D., Fogarolli, A., Sequeda, J.: Use Cases and Requirements for Mapping Relational Databases to RDF. Working draft, The World Wide Web Consortium (W3C) (2010). <https://www.w3.org/TR/rdb2rdf-ucr/>. Accessed 14 June 2016
2. Burgelman, J.-C., Osimo, D., Bogdanowicz, M.: Science 2.0 (change will happen.). First Monday **15**(7) (2010). Accessed 28 June 2016
3. Holgersen, R., Preminger, M., Massey, D.: Using semantic web technologies to collaboratively collect and share user-generated content in order to enrich the presentation of bibliographic records. Code4Lib J. **17** (2012). <http://journal.code4lib.org/articles/6695>. Accessed 20 June 2016
4. Hu, Y., Janowicz, K., McKenzie, G., Sengupta, K., Hitzler, P.: A linked-data-driven and semantically-enabled journal portal for scientometrics. In: Alani, H., et al. (eds.) ISWC 2013. LNCS, vol. 8219, pp. 114–129. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41338-4_8
5. Kalb, H., Lazaridou, P., Trier, M.: Establishing interoperability of a blog archive through linked open data. In: GI-Jahrestagung, pp. 1931–1936 (2013)
6. Latif, A., Borst, T., Tochtermann, K.: Exposing data from an open access repository for economics as linked data. D-Lib Magaz. **20**(9–10) (2014). <http://www.dlib.org/dlib/september14/latif/09latif.html>. Accessed 7 June 2016
7. Limani, F., Latif, A., Tochtermann, K.: Scientific social publications for digital libraries. In: Proceedings of 20th International Conference on Theory and Practice of Digital Libraries (2016)
8. Limani, F., Latif, A., Tochtermann, K.: Bringing scientific blogs to digital libraries. In: Proceedings of the 13th International Conference on Web Information Systems and Technologies (WebIST), Porto, Portugal, pp. 284–290, 25–27 April 2017. <https://doi.org/10.5220/0006295702840290>
9. Mahrt, M., Puschmann, C.: Science blogging: an exploratory study of motives, styles, and audience reactions. J. Sci. Commun. **13**(3), 1–16 (2014)
10. Powell, J., Collins, L., Martinez, M.: Semantically enhancing collections of library and non-library content. D-Lib Magaz. **16**(7–8) (2010). <http://www.dlib.org/dlib/july10/powell/07powell.html>. Accessed 10 June 2016
11. Passant, A., Laublet, P., Breslin, G., Decker, S.: SemSLATES: Weaving Enterprise 2.0 into the Semantic Web (2010)

12. Papadokostaki, K., Charitakis, S., Vavoulas, G., Panou, S., Piperaki, P., Papakonstantinou, A., Kondylakis, H., et al.: News articles platform: semantic tools and services for aggregating and exploring news articles. In: International Conference on Integrated Information (2016)
13. Spanos, D.-E., Stavrou, P., Mitrou, N.: Bringing relational databases into the semantic web: a survey. *Semantic Web* **3**(2), 169–209 (2012)
14. Yoose, B., Perkins, J.: The LOD landscape in libraries and beyond. *J. Libr. Metadata* **13**(2–3), 197–211 (2013)



Enhanced Querying of Open Data Portals

Mauro Pelucchi, Giuseppe Psaila^(✉), and Maurizio Toccu

University of Bergamo, Bergamo, Italy
giuseppe.psaila@unibg.it, mauro.pelucchi@tabulaex.com,
maurizio.toccu@unibg.it

Abstract. Open Data portals have become a common service provided by many Public Administrations around the world, where they openly publish many data sets concerning citizens and territories, in order to increase the amount of information made available for people, firms and public administrators. As an effect, Open Data corpora has become so huge that it is impossible to deal with them by hand; as a consequence, it is necessary to develop tools for effectively querying corpora of open data, in order to find the desired data sets.

In our previous work [1], we presented a novel technique to query open data corpora. In this paper, we present an evolution of that technique, obtained by refining some steps and by introducing some novelties. We still rely on the *blindly querying* approach: the user does not have to know in advance the actual structure of possibly thousands of data sets, but formulates the query trying to characterize the items of interests; in fact, a novelty of our approach is that our technique looks for single items within data sets, not for data sets. Then, the technique tries to rewrite the query by exploiting the catalog of the corpus in order to find the most similar and relevant terms.

The main enhancement introduced in the technique and presented in this paper is the way the technique looks for similar terms in the catalog, that now is based on a semantic approach: the *WordNet* dictionary is exploited to get synonyms of terms in the query. Furthermore, a new set of experiments has been performed, in order to prove the effectiveness of the enhanced technique.

1 Introduction

Citizens looking for information concerning territories and citizenship now consider obvious to look for them in an *Open Data Portal*, i.e., a web site in which Public Administrations publish data sets concerning their activities and citizens. Since these data sets are open, i.e. publicly available for any interested people and organizations, they are called *Open Data*.

Data sets published by Open Data Portals are *Structured* data sets: in the simplest form, they are CSV files with a fixed number of fields, with field names reported in the first line. Alternatively, they are often represented as vectors of JSON objects, without nesting (they must be represented in a flat way), and (less frequently) as structured XML documents (without exploiting nesting

capabilities of XML). In practice, data sets are always thought as tables: an item in the data set (CSV row, JSON object, XML element) represents a table row.

The hugeness of Open Data corpora published by open data portals asks for tools to effectively query Open Data corpora. In particular, it is not possible to think that a user knows the actual structure of thousands of data sets in a corpus: he/she queries the corpus in a *blind way*; then, it is responsibility of the engine to adapt the query to the structure of data sets actually published in the corpus.

In our previous work [1], we proposed a technique to query a corpus of open data, that exploits the *blindly querying* approach. In particular, the technique takes the original query and rewrites it, by looking for similar terms that are actually present in the structure of data sets and can be effective in getting the desired data sets. Furthermore, an important novelty of our technique is that it looks for single items within data sets, i.e., items that satisfy a selection condition specified in the query by the user.

For example, a user might want to get information about high schools located in a given city named “My City”, being interested in their *name*, *address* and *reputation*. The query mechanism should be able to focus the search on those data sets that actually contain the items of interest, and retrieve them. However, at the same time, it should be able to *extend the search* to those data sets that could potentially contain items of interest, by considering similar names in the corpus catalog (i.e., the list of data sets in the corpus and their fields). For example, a data set with field “city_name” or simply “city” in place of “cityname”.

In this paper, we proposed an enhanced version of the technique proposed in [1]. Apart from minor improvements, obtained by reorganizing some processing steps, the major improvement concerns the way queries are rewritten. In fact, given a term, we look not only for lexicographic similar terms present in the catalog, but also for semantically similar terms: we exploit the *WordNet* dictionary to get synonyms. This way, our technique is able to better explore the corpus, fostering the *blindly querying* approach.

The new version of the technique has been implemented in the *Hammer* prototype and validated by means of an experimental campaign.

The paper is organized as follows. Section 2.1 gives some preliminary definitions. Section 2.2 formally defines the problem we are dealing with. In Sect. 3, the overall query process is presented, presenting the global view of the technique. Next sections presents the technique in details: Sect. 4 presents the main innovation of this paper, i.e., the identification of alternative terms for terms in the query, on the basis of semantic similarity (synonyms in *WordNet* dictionary) and lexicographic similarity; Sect. 5 explains the steps that lead to obtain rewritten queries; Sects. 6 and 7 present the core of the technique, i.e., extraction of relevant keywords from within rewritten queries and the final steps of the technique, where rewritten queries are actually used to find out data sets and relevant items are extracted from within them. The experimental evaluation is

shown in Sect. 8, while Sect. 9 discusses some related works. Section 10 draws the conclusions.

2 Problem Statement

In this section, we precisely state the problem. To do that, we need to introduce some preliminary concepts.

2.1 Preliminary Definitions

Definition 1: Data Set (From [1]). An *Open Data Set* ods is described by a tuple

$$ods : \langle ods_id, dataset_name, schema, metadata \rangle$$

where ods_id is a unique identifier, $dataset_name$ is the name of the data set (not unique); $schema$ is a set of fields names, $metadata$ is a set of pairs ($label, value$), which are additional meta-data associated with the data set. \square

Definition 2: Instance and Items (From [1]). With *Instance*, we denote the actual content of data sets. It is a set of *items*, where an item is a row of a CSV file or data table, as well as a flat object in a JSON vector. \square

Many open data portals provide open data sets in several formats. Commonly, they adopt CSV (comma separated values), but it is becoming frequent the adoption of JSON (JavaScript Object Notation) and XML.

The adoption of JSON is motivated by the fact that, now, it has become the de-facto standard for data sharing in the world of Web Services. A JSON file is easy to read, much lighter than XML and JavaScript (the programming language used for developing client-side web applications) natively interprets JSON data.

However, in the open data world, it is used only as an alternative representation for flat tabular data: a CSV file can always be transformed into a vector of JSON objects, where all objects in the vector have the same structure, i.e., the same fields and fields have only simple values (there are not nested objects).

Consequently, JSON capabilities to describe nested complex structures, as well as collections of heterogeneous objects, are not exploited in this context.

To illustrate, consider Fig. 1. We report a fragment of the open data set named *Local Authority Maintained Trees* published on the *London Data Store*, that is the official open data portal of London City. Notice how a flat CSV file (in the upper part of the figure) can be easily transformed into a vector of JSON objects (lower part of the figure). As the reader can see, the price to pay is that the JSON version is more verbose, since field names must be repeated for each object. However, in the optic of integrating the items extracted from several open data sets, having different structures, into a unique collection of results, JSON is the natural representation format for our tool.

CSV Version

```
ID,Borough,Species name,Common name,Display name,Ox,Oy
1,Barking ,ACER PSEUDOPLATANUS 'BRILLIANTISSIMUM' , ,
    Maple,548320.8,189593.7
2,Barking ,TAXUS BACCATA FASTIGIATA,,Other,548297.9,189590.2
```

JSON Version

```
[
  {
    "ID": 1,
    "Borough": "Barking ",
    "Species name": "ACER PSEUDOPLATANUS 'BRILLIANTISSIMUM' ",
    "Common name": "",
    "Display name": "Maple",
    "Ox": 548320.8,
    "Oy": 189593.7
  },
  {
    "ID": 2,
    "Borough": "Barking ",
    "Species name": "TAXUS BACCATA FASTIGIATA",
    "Common name": "",
    "Display name": "Other",
    "Ox": 548297.9,
    "Oy": 189590.2
  }
]
```

Fig. 1. Fragment from the data set *Local Authority Maintained Trees* published in the London Data Store.

Definition 3: Global Meta-Data (From [1]). The list of fields of items in the open data set *ods* is *ods.schema*. We call the triple $\langle \text{data_set_name}, \text{schema}, \text{metadata} \rangle$ the *Global Meta-Data* of the data set. \square

Definition 4: Corpus and Catalog (From [1]). With $C = \{ods_1, ods_2, \dots\}$ we denote the corpus of *Open Data sets*. The *catalog* of the corpus is the list of descriptions, i.e., global meta-data, of open data sets in C (see Definition 1). \square

At this point, we can introduce the concept of *Query*, to illustrate the approach we propose to query an open data corpus.

Definition 5: Query (From [1]). Given a *Data Set Name* *dn*, a set P of field names (properties) of interest $P = \{pn_1, pn_2, \dots\}$ and a selection condition *sc* on field values, a query q is a triple $q: \langle dn, P, sc \rangle$. \square

As the reader can see, a section condition must be specified in the query: this way, the user can narrow his/her search only to items of interest.

Now, we define a concept that will be very useful throughout the paper.

Definition 6: Query Term (From [1]). With *Query Term* (*term* for simplicity) we denote a data set name $q.dn$, a field name appearing in $q.P$, in $q.sc$, a constant appearing in $q.sc$. \square

Note that a term could be also compound by many words, in general two words, separated by blank spaces, underscores or lines. In fact, property names in open data sets can contain blanks, underscores and lines.

2.2 Problem Definition

Consider an open data corpus published by an *Open Data Portal*. Queries allow users to retrieve, from within the corpus C , those data sets, possibly having the specified name $q.dn$ or a similar one, that contain items (e.g., rows or JSON objects) with the desired field names $q.P$ (or similar) and can be actually filtered on the basis of the selection condition $q.sc$, so that users can obtain filtered items. By relying on the definition of a *relevance measure* for data sets (in Sect. 7), we can define the overall problem. However, we first of all report a statement, that explain what properties the relevance measure must provide.

Statement 1: Relevance Measure. Consider an open data set ods_i . We need to define a *Relevance Measure* $rm(ods_i) \in [0, 1]$, suitable to evaluate the relevance of the data set w.r.t. the query. Since we rely on the *blindly querying* approach, the relevance measure must provide the following features:

1. An open data set could be relevant for the query even if does not contain all terms specified in the query;
2. An open data set could be relevant for the query even if it contains terms that are lexicographically similar to the ones specified in the query;
3. An open data set could be relevant for the query even if it contains terms that are semantically similar (synonyms) to the ones specified in the query. \square

The previous statement states what we expect from a relevance measure. It will be defined later in the paper. On this basis, we can define the problem.

Problem 1 (From [1]): Given a *Relevance Measure* $rm(ods) \in [0, 1]$ of a data set $ods \in C$, and a minimum threshold $th_{rm} \in [0, 1]$, a data set ods is *relevant* if $rm(ods) > 0$ and $rm(ods) \geq th_{rm}$.

The result set $RS = \{o_1, o_2, \dots\}$ of query q is the set of items (rows or JSON objects) o_i such that ods_j is a relevant data set, $o_i \in Instance(ods_j)$ and o_i satisfies the selection condition $q.sc$. \square

Example 1: The sample query about schools in Sect. 1 becomes:

$$\begin{aligned} q :< dn &= [\text{Schools}] \\ P &= \{[\text{Name}], [\text{Address}], [\text{Reputation}]\}, \\ sc &= ([\text{City}] = \text{"My City"} \text{ AND } [\text{Type}] = \text{"High School"}) > . \end{aligned}$$

Based on Statement 1, we expect to get items in data sets having field $[\text{Name}]$, or field $[\text{School Name}]$ in its place (lexicographic similarity), as well as field $[\text{Address}]$ or, in its place, $[\text{Street}]$ (semantic similarity). \square

3 Processing Steps

The contribution of this paper is a novel technique to solve Problem 1. The technique moves from the one previously presented in [1], but modified and enriched with semantics.

The proposed technique is built around a query mechanism based on the Vector Space Model [2], encompassed in a multi-step process devised to deal with the high heterogeneity of open data sets and the blindly query approach (the user does not know the actual schema of data sets in the corpus). Recall that, in our vision, the user performs a *blindly query*, but asks for specific field names and for items (rows or JSON objects) that satisfy precise selection conditions. The system is fully aware of data sets schemata, but due to their heterogeneity it is not possible to rely only on exact matching. Thus, it is necessary to focus the search, identifying those terms in the query that mostly represent the query essence and the capability of the query to select items within data set instances. At the same time, it is necessary to expand the search space, considering terms which are similar to those in the query, in order to address the fact that users are unaware of actual schemata of data sets.

Moving from the above considerations, we defined the *steps* performed by our technique.

- Step 1: *Term Extraction and Retrieval of Alternative Terms*. The set $T(q)$ of terms is extracted from within the query q . Then, for each term $t \in (q)$, the set $Alt(t)$ of similar terms is build. A term $t' \in Alt(t)$ either if it is lexicographically similar (based on a string-similarity measure) or semantically similar (synonym) based on *WordNet* dictionary, or a combination of both. Terms in $Alt(t)$ are actually present in the catalog of the corpus.
- Step 2: *Neighbour Queries*. For each term $t \in T(q)$, alternative terms $Alt(t)$ are used to to derive, from the original query q , the *Neighbour Queries*, i.e., queries which are similar to q . Both q and the derived neighbour queries are in the set Q of queries to process.
- Step 3: *Keyword Selection*. For each query to process $\bar{n}q \in Q$, keywords $K(\bar{n}q)$ are selected from terms in $\bar{n}q$, in order to find the most representative/informative terms for finding potentially relevant data sets.
- Step 4: *VSM Data Set Retrieval*. For each query $\bar{n}q \in Q$, the selected keywords $K(\bar{n}q)$ are used to retrieve data sets based on the Vector Space Model [2] approach: in this way, the set of possibly relevant data sets is obtained.
- Step 5: *Schema Fitting*. The full set of field names in each query $\bar{n}q \in Q$ is compared with the schema of each selected data set, in order to compute the relevance measure rm . Data sets whose schema better fits the query will be more relevant than other data sets.
- Step 6: *Instance Filtering*. Instances of relevant data sets (i.e., with relevance measure greater than or equal the minimum threshold th_{rm}) are processed in order to filter out and keep only the items (rows or JSON objects) that satisfy the selection condition.

In the next sections, we will discuss each processing step in details.

4 Retrieving Alternative Terms

Following Statement 1 (motivated by the *blindly querying* approach), for each term in the query we have to identify a pool of alternative terms, on the basis of semantic similarity, lexicographic similarity or both. The retrieval of semantically similar terms is the main novelty introduced in this paper, w.r.t. paper [1].

4.1 WordNet and Synonyms

The major innovation w.r.t. [1] is the semantic enrichment based on *WordNet* [3]. The idea is that alternative terms of t are synonyms of t , in addition to similar terms.

However, the application of this idea is complicated by the fact that property names in data sets could not be simple words, but words attached each other with or without some characters such as blanks, lines or underscores.

First of all, let us define the basic concept of *WordNet SynSet*.

Definition 7: WordNet Synset. Given a term t , function $WNSynSet(t)$ returns the set of synonyms of term t , based on *WordNet* dictionary. Note that $WNSynSet(t) = \emptyset$, either if t does not exist or if t have no synonyms. \square

On this basis, we can further generalize, in order to capture compound terms. Suppose that a term t is formed by two terms t' and t'' . The intuition is that the synset of t should be the Cartesian product $WnSynSet(t') \times WnSynSet(t'')$. The following definition precisely defines the concept.

Definition 8: Synset for Compound Terms. Consider a term t and replace underscores and lines with blanks, obtaining the blanked term t' . Two situations may occur.

1. *Explicit Compound Term.* If t' contains more than one term, i.e., $r' = t_1 \bullet t_2 \bullet \dots \bullet t_n$ (the “ \bullet ” is the string composition operator), the synset for the compound term is

$$\begin{aligned} \text{CompoundSynSet}(t) &= ((\{t_1\} \cup WNSynSet(t_1)) \times (\{t_2\} \cup WNSynSet(t_2))) \\ &\times \dots \times (\{t_n\} \cup WNSynSet(t_n)) - \{t'\} \end{aligned}$$

i.e., for each term t_i , we look for the synset from *WordNet* and combine them with a Cartesian product (since original terms are considered too, the original blanked term t' could be obtained and must be removed).

2. *Implicit Compound Term.* If term t does not contain blanks, underscores and lines, we try to discover implicit compound terms.

Consider the length $l = |t| > 0$ of term t . With ls_i we denote the left substring of length i , while with $rs_{(l-i)}$ we denote the corresponding right substring of length $l - i$.

$$\begin{aligned} \text{CompoundSynSet}(t) &= \cup_{i=1 \dots (l-1)} ((\{ls_i\} \cup WNSynSet(ls_i)) \\ &\times (\{rs_{(l-i)}\} \cup WNSynSet(rs_{(l-i)})) - \{ls_i \bullet rs_{(l-i)}\}) \end{aligned}$$

i.e., we extract any pair of substrings from t and look for their synsets in *WordNet*. Notice that the final set could be empty. \square

Now, we can aggregate all possible synonyms into the set of *Preliminary Alternative Terms*.

Definition 9: Preliminary Alternative Terms. Given a term t , the set of *Preliminary Alternative Terms* is obtained by uniting all possibly synonyms and the term t itself: $\overline{Alt}(t) = \{t\} \cup WNSynSet(t) \cup CompoundSynSet(t)$. \square

Notice that term t is an alternative term of itself, so that it can be natively considered for lexicographic similarity.

4.2 Lexicographic Similarity

Once the preliminary set of alternative terms has been computed, it is necessary to assess it against the catalog of the corpus. In fact, terms (included the original one) may not be present in the catalog, so it is necessary to replace them with similar terms that are actually present in the catalog. To this end, we have to rely on a *lexicographic* string similarity measure. In this context, one of the most feasible lexicographic similarity functions is the *Jaro-Winkler* measure (see [4,5]), because it gives more importance to strings with a common prefix: given two terms t and t' , $sim(t, t') \in [0, 1]$ (if 1, t and t' coincides; if 0, t and t' are completely different).

Definition 10: Top-most Similar Terms. Consider a term t . Given an integer number Max_Alt and a minimum threshold $th_sim \in [0, 1]$ for lexicographic similarity, $TopMost(t, Max_Alt, th_sim) = \{t_1, \dots, t_n\}$ (with $n = Max_Alt$) is the set of Max_Alt top-most terms w.r.t. $sim(t, t_i)$ that are actually present in the catalog of the corpus, such that $sim(t, t_i) \geq th_sim$. \square

The idea is that we do not consider all possible similar terms, but only the most similar ones actually present in the catalog of the corpus.

Anyway, potentially alternative terms must be weighted, in order to measure their usefulness as alternative terms.

Definition 11: Weight of Alternative Terms. Given a term t , consider the set of preliminary alternative terms $\overline{Alt}(t)$ and a term t' . The weight of term t' as an alternative term for t , denoted as $wat_t(t')$, is defined as follows.

- If $t' \in \overline{Alt}(t)$ and t' is actually present in the catalog, $wat_t(t') = 1$; if t' is not present in the catalog, it is $wat_t(t') = 0$.
- If $t' \notin \overline{Alt}(t)$ and it is actually present in the catalog, $wat_t(t') = sim(\bar{t}, t')$, where $\bar{t} \in \overline{Alt}(t)$ with highest lexicographic similarity. If t' is not present in the catalog, it is $wat_t(t') = 0$. \square

At this point, we can specify which terms are actually considered alternative for a given term t in the query q .

Definition 12: Alternative Terms. Consider a term $t \in T(q)$ and the set of preliminary alternative terms $\overline{Alt}(t)$. The set of *Alternative Terms* $Alt(t)$ for term t is defined as follows:

$$Alt(t) = \{t' | ((t' \in \overline{Alt}(t) \vee (t' \in TopMost(\bar{t}, Max_Alt, th_sim) \wedge \bar{t} \in \overline{Alt}(t))) \wedge wat_t(t') > 0)\} \quad \square$$

In other words, we consider as alternative terms for a query term t , the term t itself if present in the catalog, the synonyms that are actually present in the catalog and all terms, actually present in the catalog, as much as possible lexicographically similar either to t or to a synonym. Parameters Max_Alt and th_sim can be varied to tune the the effectiveness.

5 Deriving Neighbour Queries

Once for each term $t \in T(q)$ the set $Alt(t)$ of alternative terms is derived, we can perform the query rewriting phase, obtaining the so-called *Neighbour Queries*.

Definition 13: Neighbour Query. Consider the original query q , with its term vector $T(q) = [t_1, \dots, t_n]$ and a vector of weights $W(q)$, where $W(q)[i] = 1$, for each term $t_i = T(q)[i]$. A neighbour query pnq , with its term vector $T(pnq)$ and its weight vector $W(pnq)$, is obtained as follows:

- $|T(pnq)| = |T(q)|$.
- One or more terms $t_i \in T(q)$ are replaced with one term chosen from the set of alternative terms, $t_j^i \in Alt(t_i)$. If t_i is not replaced, $T(pnq)[i] = t_i$; otherwise, $T(pnq)[i] = t_j^i$. Correspondingly, the new potential neighbour query pnq is obtained by performing the same replacements in the text of query q .
- The vector of weights for the potential neighbour query is as follows: if t_i is not replaced, $W(pnq)[i] = 1$; otherwise, $W(pnq)[i] = wat_{t_i}(t_j^i)$. \square

Any query obtained by replacement of one or more terms is a potential neighbour query. However, their number could be high and the query could be possibly too far away from the original query; therefore, we have to prune those potential neighbour queries that may be too distant, based on the cosine similarity metric applied to the weight vectors.

Definition 14: Queries to Process. Given the original query q , for each potential neighbour query p , p we compute the $CosineSim(W(q), W(p))$.

The set Q of (neighbour) queries to process contains the Max_N (integer number) potential neighbour queries with top-most cosine similarity values. \square

Note that not necessarily the original query q is present in Q : in fact, the absence of some terms in the catalog can cause that it is not obtained as neighbour query.

Also notice that Max_N is a parameter that can be changed to tune effectiveness of retrieval. In our experiments, we set it to 10: this way, we avoid the combinatorial explosion of queries to process.

6 Keyword Selection

Consider a query $nq \in Q$. Terms appearing in nq are not equally important/informative for the query. If the set of terms used to retrieve possibly relevant data sets contains terms that are not important or much less important than others, in Step 4 we can retrieve data sets that may be not actually relevant for the query. Thus, we adopted a *keyword selection technique* that extracts, from a query $nq \in Q$, the set of (meaningful) keywords $K(nq)$.

The technique we adopted in [1], that hereafter we report, was inspired by the technique introduced in [6] for the context of classical search engines. However, we introduced several variations (structure of the graph and scores) to cope with peculiarities of our problem.

The keyword selection technique proceeds as follows (from [1]):

1. The *Term Graph* is built, where each node corresponds to a term in the query nq and an edge connects two terms and weights their relationship (see Fig. 3 for a sample graph discussed after the keyword selection algorithm reported in Fig. 2).
2. An iterative algorithm:
 - (a) Chooses the most relevant term.
 - (b) Modifies the weights of not selected terms, propagating a decreasing reduction of informativeness to neighbours of the selected term.
3. Only most informative terms are kept and inserted into the keywords set $K(nq)$.

Term Graph Construction. Each node in the *Term Graph* corresponds to a term in the query. There are three types of nodes: one *Data Set Name Node*, corresponding to the data set name $nq.dn$; *Field Nodes*, corresponding to field names in $nq.P$ or in $nq.sc$; *Value Nodes*, corresponding to constants appearing in comparison predicates in $nq.sc$.

Nodes are connected each other by an edge if terms are related to each other. In particular: a field node is connected with the data set name node; two field nodes are connected if they are the operands of the same comparison predicate in $nq.sc$; a field node and a value node are connected if the field (described by the field node) is compared with the constant value (described by the value node) by a comparison predicate in $nq.sc$.

Each node (term) has two scores: a *representativeness* score, denoted as r_score , and an *informativeness* score, denoted as i_score .

Definition 15: Representativeness Score (From [1]). The representativeness score r_score denotes the capability of the term to find a relevant data set, i.e., how much it represents a specific subset of data sets. In particular, we consider two components for representativeness: *Intrinsic Representativeness* and *Extrinsic Representativeness*. The former is related to the presence of the term in the query; we denote it as $irep(t)$ (where t is a term). The latter is related to the presence of the term in global meta-data of data sets, thus it measures

the actual capability of the term to focus the search; we denote the extrinsic representativeness as $erep(t)$. Both $irep$ and $erep$ are defined in the range $[0, 1]$.

The r_score of a term is obtained by combining the intrinsic and the extrinsic representativeness, by means of the average of them, i.e., $r_score = (irep(t) + erep(t))/2$. This way, both contribute and balance each other. \square

Intrinsic Representativeness. Depending on the role played by a term in the query, we established different values for the intrinsic representativeness.

- The intrinsic representativeness of the *Data Set Name* is $irep(t) = 1.0$ if there exists a data set $nq.dn$ in the corpus C ; otherwise, $irep(t) = 0.0$.
 - The intrinsic representativeness of a *Field Name* is $irep(t) = 0.5$ if there exists at least a data set that has a field named t ; otherwise, $irep(t) = 0.0$.
- The intrinsic representativeness for string value nodes and date value nodes, is $irep(t) = 0.5$, while it is $irep(t) = 0.2$ for numerical value nodes.

Extrinsic Representativeness. The approach we adopted to define the extrinsic representativeness, is inspired by the *idf* (*inverse document frequency*) measure [2]. Its classical definition is $idf = \log(D/d(t))$, where D is the total number of documents, while $d(t)$ is the number of documents that contain term t . The lower the number of documents that contain t , the higher the value of idf , because term t is more capable to focus the search on relevant documents.

Moving from this idea, we defined the extrinsic representativeness in order to be in the range $[0, 1]$ and to be suitable for our context.

Definition 16: Normalized Data Set Frequency (From [1]). Consider the corpus C of open data sets and its cardinality $|C|$. Consider a term t , the set $Matched(t)$ of data sets that contain term t in global meta-data (such that $Matched(t) \subseteq C$) and its cardinality $|Matched(t)|$.

The *Normalized Data Set Frequency* of term t is defined as

$$ndf(t) = \log_2 \left(\frac{|Matched(t)|}{|C|} + 1 \right)$$

that is defined in the range $[0, 1]$.

If term t is contained in the global meta-data of all data sets, $ndf(t) = 1$ (the argument of the logarithm is 2). If term t is not contained in any global meta-data, $ndf(t) = 0$, because the argument of the logarithm is 1. \square

Definition 17: Extrinsic Representativeness (From [1]). Given a term t and its normalized data set frequency $ndf(t)$, its *Extrinsic Representativeness* is defined as:

$$\begin{aligned} erep(t) &= 1 - ndf(t) && \text{if } ndf(t) > 0 \\ erep(t) &= 0 && \text{if } ndf(t) = 0 \end{aligned}$$

\square

The rationale is the following: if a term is very frequent, its extrinsic representativeness is close to 0, because it is not able to focus the search. In contrast, not frequent terms are able to focus the search, and their extrinsic representativeness is close to 1.

Definition 18: Informativeness Score (From [1]). The informativeness score i_score measures the degree of information that could be obtained by selecting a term as keyword. At the beginning, an initial value is set: we chose $i_score=1.0$ for data set name nodes and field nodes that appear in the selection condition $nq.sc$. We chose $i_score=0.8$ for value nodes and field nodes appearing only in $nq.P$. In fact, constants are naturally less informative than data set names and field names; furthermore, they are less likely to appear in meta-data of data sets, thus the data set retrieval step based on VSM would result ineffective. The same is for field names not appearing in the selection condition: they are less informative than those appearing in the selection condition.

During the keyword selection process, the selection of a term affects the informativeness of neighbours: in fact, selecting two neighbours may be redundant. Consequently, the procedure will modify the i_score values of not selected terms. \square

Keyword Selection Algorithm. We now report the keyword selection algorithm, that we introduced in [1]. The algorithm is inspired by the algorithm reported in [6], but modified because, in our term graph, edges are neither

```

Procedure KeywordSel
Input:  $N$  set of nodes,  $E$  set of edges
a node is  $n = \langle t, i\_score, r\_score, marked = false \rangle$ 
Output: the set of selected keywords  $K$ 
begin
1.  $K = \emptyset$ ; // set of selected keywords
   do
2.    $max\_score = 0.0$ ; //initial max score
3.    $max\_node = null$ ; //no node selected
4.   for each node  $n \in N$  with  $n.marked = false$  do
5.      $score = n.i\_score + n.r\_score$ ;
6.     if  $score > max\_score$  then
7.        $max\_score = score$ ;
8.        $max\_node = n$ ;
9.     end if
10.  end for each
11.  if  $n \neq null$  and  $max\_score > 1.0$  then
12.     $K = K \cup \{n.t\}$ ; //term selected as keyword
13.     $n.marked = true$ ; //node marked as selected
14.    call procedure UpdateIScore( $N, E, n$ );
15.  end if
16.  while  $max\_score > 1.0$ ;
end procedure

```

Fig. 2. Procedure *KeywordSel*, that selects possibly relevant keywords (introduced in [1]).

labeled nor weighted. The main procedure of the algorithm, named *KeywordSel*, is reported in Fig. 2.

The algorithm works on the input sets of nodes and edges; notice that nodes have an additional field named *marked*, denoting selected nodes when *true*.

The main cycle from lines 2 to 13, at each step selects the node not yet selected with the highest value for $i_score + r_score$, provided that the sum is greater than 1. This is because below this threshold the capability of retrieving significant data sets becomes too low. The *for* loop from line 4 to line 8 looks for the highest $i_score + r_score$ value (and the corresponding node) in not marked nodes. Then, the *if* instruction at line 9 verifies that the selected node (if any) has the sum $i_score + r_score$ greater than 1: if so, the node is marked as selected, the corresponding term is inserted into the keyword set K and the procedure *UpdateIScore* is called (line 12) to propagate a reduction of informativeness.

Procedure *UpdateIScore* performs the update of the i_score , by propagating from the selected node. The propagation mechanism can be described in a recursive way.

- Starting from the selected node s , for each node z' connected with s , the reduction factor is $rf(z') = s.r_score / 2/n$, where n is the number of nodes connected with s .
- Consider a node z that just received a reduction factor $rf(z)$ and the set $Out(z)$ of nodes connected to z that have not received yet a reduction factor. For each node $z' \in Out(z)$, the reduction factor is $rf(z') = rf(z) / 2 / |Out(z)|$. The propagation continues until all nodes have a reduction factor.
- For each node z , the i_score is decreased by the reduction factor $rf(z)$, i.e., $z.i_score = z.i_score - rf(z)$ (where the lower bound is $z.i_score = 0$).

This way, the representativeness of the selected node decreases the informativeness of neighbours; the reduction becomes lower and lower as the distance from the selected node increases.

Figure 3a reports the initial term graph for the query in Example 1. Figure 3b shows how i_score values change, showing the propagation from the selected node (*City*) to the other nodes.

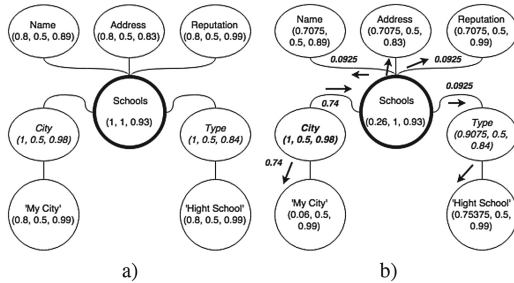


Fig. 3. Initial state of the Term Graph (a) and after the first keyword selection (b). The triple in each node is $(i_score, irep, errep)$ (presented in [1]).

7 Data Set Retrieval

Each query $nq \in Q$ has associated the vector $K(nq)$ of keywords selected in step 3 (Sect. 6). Now, in step 4, the Vector Space Model [2] is applied to retrieve possibly relevant data sets.

The VSM approach relies on the concept of *inverted index* created on the bases of terms in the global meta-data of each data set. Definitely, in our context the inverted index IN implements a function $f(t) \rightarrow \{ods_id_1, ods_id_2, \dots\}$, that, given a term t , returns the set of data set identifiers whose global meta-data contain term t .

Consider a vector of keywords $K(nq) = [k_1, \dots, k_n]$. Then, given a data set identifier ods_id , the occurrences of keywords associated with ods_id in the inverted index IN are represented by the vector $m(ods_id) = [p_1, \dots, p_n]$, where $p_j = 1$ if the data set is associated with the keyword, $p_j = 0$ otherwise.

The *Keyword-based Relevance Measure* $krm(ods, nq)$ of a data set ods for a query $nq \in Q$ is the average of p_j values. The maximum value of $krm(ods)$ is 1, obtained when the data set is associated with all the keywords, while values less than 1 but greater than a minimum threshold th_krm , which is a parameter of the technique.

7.1 Schema Fitting

Step 5 of our technique computes, for each data set retrieved in Step 4, the final relevance measure rm , by composing the keyword-based relevance measure krm of a data set with the *fitting degree* of its schema w.r.t. the field names in the query. This step is necessary to identify, among all retrieved data sets, those having an instance that can be effectively processed by the selection condition $nq.sc$.

We adopt an approach based on a weighted inclusion measure, where the weight of each field name depends on the role played in the query by the field name.

Definition 19: Schema Fitting Degree (From [1]). Consider a data set ods and its schema $ods.schema = \langle pn_1, pn_2, \dots \rangle$. Then, consider a query $nq \in Q$ and the set of field names $N = \{n_1, \dots, n_k\}$ that appear in the selection condition $nq.sc$ and/or in the list of field names of interest $nq.P$. For each name $n_i \in N$, a weight $w_i \in [0, 1]$ is associated (see Definition 20), depending on the role played by n_i in the query. The *Schema Fitting Degree* $sfd(ods, nq)$ is defined as:

$$sfd(ods, nq) = \frac{\sum_{i=1}^{|N|} w_i \times p(n_i)}{\sum_{i=1}^{|N|} w_i}$$

where function $p(n_i) = 1$ if there exists a field name $pn_j = n_i$ in $nq.schema$, $p(n_i) = 0$ otherwise. \square

The following definition defines how names are weighted.

Definition 20: Weight (From [1]). The weight w_i of a field name $n_i \in N$ is

$$w_i = \max(pw_i, scw_i)$$

where pw_i is the weight due to the presence of n_i in the list $nq.P$ of fields of interest in the query, while scw_i is the weight due to the presence of n_i in the selection condition $nq.sc$.

- $pw_i = 0.5$ if field name $n_i \in nq.P$ (may be relevant to have this field in the resulting items); $pw_i = 0$ otherwise.
- $scw_i = 1$ if the selection condition $nq.sc$ contains one single predicate with n_i , or the predicates in the condition are ANDed each other; $scw_i = 0.7$ if predicates are ORed each other. \square

Finally, we define the relevance measure for a data set.

Definition 21: Data Set Relevance Measure (From [1]). Consider a data set ods and a query $nq \in Q$. Given its *Keyword-based Relevance Measure* $krm(ods, nq)$ and its *schema Fitting Degree* $sfd(ods)$, the *Relevance Measure* of data set ods w.r.t. query nq is

$$\begin{aligned} &rm(ods, nq) \\ &= (1 - \alpha) \times krm(ods, nq) + \alpha \times sfd(ods, nq) \end{aligned}$$

where $\alpha = 0.6$. The *data set relevance measure* $rm(ods)$ is

$$rm(ods) = \max(rm(ods, nq_i)),$$

for each $nq_i \in Q$. \square

The rationale is that the schema fitting degree corrects the keyword-based relevance measure. The reason is simple: the keyword-based relevance measure is obtained by querying the inverted index in a blind mode, i.e., irrespective of the fact that a keyword is a data set name, a field name or a word appeared in meta-data. The keyword-selection mechanism partially corrects this blind approach, to discard those data sets that could result accidentally relevant.

Nevertheless, in order to perform item selection from downloaded data sets, how the field names in the query match the actual schema of data sets becomes crucial, because it is impossible to evaluate the selection condition if fields in the condition are not present (all or partially) in the schema of the data sets. This also motivates the choice of $\alpha = 0.6$.

Finally (see Sect. 2.2), the minimum threshold th_rm is used to discard less relevant data sets and keep only those such that $rm(ods) \geq th_rm$.

The process concludes with Step 6, where instances of selected data sets are downloaded from the portal; then, each data set instance is filtered, in order to extract only items (CSV rows or JSON objects) that actually match the neighbour query $nq \in Q$ that best fits with the data set schema.

Selected items constitute the result set of the process (see Problem 1 in Sect. 2.2).

8 Experimental Evaluation

We implemented our technique in a prototype named *Hammer* (see [7], for a description of the prototype) and evaluated the effectiveness of the technique.

In [1], we performed experiments on a set of 2301 open data sets published by the *Africa Open Data* portal¹. That portal was chosen due to the high heterogeneity of data sets in terms of themes and languages. However, for the enhanced technique we are describing in this paper, enriched with semantics provided by *WordNet* dictionary, we considered only English language. Therefore, experiments were done on a different Open Data portal, i.e., the *New York City* portal². This portal publishes 1711 data sets, ranging from data concerning territory to the list of taxi drivers with temporarily-suspended license. In particular, this latter type of data sets continuously vary, so, in order to guarantee the repeatability of experiments, we had to carefully formulate queries, in order to look for stable data sets.

We performed 6 different queries. They are reported in Fig. 4. Hereafter, we describe them.

- In query q_1 , we look for data concerning water consumption in year 2015 and year 2016. We are interested in properties named *year*, *population* (served

```

q1:<dn=[water],
    P={ [year], [population], [consumption], [percapita] },
    sc=( [year] = 2015 OR [year]=2016 ) >
q2 :<dn=[performance],
    P={ [indicator], [month], [report] },
    sc=( [indicator] = "Public Contacts" OR
        [indicator]="Daily Average - Crimes Against Person" OR
        [indicator]="Daily Average - Crimes Against
Property" ) >
q3 :<dn=[fountains],
    P={ [property], [site], [drinking], [borough] },
    sc=( [borough] = "M" ) >
q4 :<dn=[healthcare],
    P={ [borough], [facility_name], [location], [phone] },
    sc=( [facility.type]="Child Health" AND [borough]="Bronx" ) >
q5 :<dn=[results],
    P=
        { [date], [fecalcoliform], [averageturbidity],
        [turbidity] },
    sc=( [date] >= "2016-01-01T00:00:00" ) >
q6 :<dn=[leaks],
    P={ [leaking_fixture], [leak_type],
        [water_wasted], [total_cost], [location] },
    sc=( [leaking_fixture] = "faucets" ) >

```

Fig. 4. Queries for the experimental evaluation.

¹ <https://africaopendata.org/>.

² <https://data.cityofnewyork.us>.

- by the water provider), *consumption* (of water in the year) and *per capita consumption* in the year.
- In query q_2 , we look for performance indicators from 911, the emergency phone service in USA. In particular, we are interested in monthly reports concerning generic public contacts, or daily average calls for crimes against persons or for crimes against property. Notice that the name of the desired data set is *performance*, since we wanted to specify a very generic data set name, to stress the capabilities of the technique.
 - In query q_3 , we look for data concerning the state of fountains in the borough named M , being interested in the location of the fountain and if the dispensed water can be drunk.
 - In query q_4 we look for information about health-care facilities in *Bronx* borough.
 - In query q_5 , we look for data concerning bacteriological analysis of water in the city water distribution network, since January 1, 2016 (notice that the date is written as "2016-01-01T00:00:00", to stress the similarity search).
 - Finally, query q_6 looks for data about leaks in city water distribution networks, concerning faucets, in order to evaluate the cost of the leaks.

To perform the evaluation, we downloaded all the data sets and we explored and manually labeled every data set and item as correct or wrong w.r.t. each query³.

We performed 4 tests, in order to evaluate the effects of varying some parameters. Hereafter, we recall the parameters and report the values we adopted for them.

- *Max_N*: the max number of neighbour queries to select to process is set to 10.
- *Max_Alt*: the maximum number of similar alternative terms is set to 3.
- *th_sim*: for the minimum threshold for lexicographic similarity, we chose two distinct values, i.e., 0.8 and 0.9.
- *th_krm*: for the minimum threshold for *Keyword-based Relevance Measure*, we considered two different values, i.e., 0.2 and 0.3.
- *th_rm*: for the threshold for the global relevance measure, we considered two different values, i.e., 0.2 and 0.3.

Not all the combinations of varying parameters were considered, to simplify the analysis. We defined 4 configurations (tests).

- For *Test_1*, $th_sim = 0.9$, $th_krm = 0.3$ and $th_rm = 0.3$;
- For *Test_2*, $th_sim = 0.9$, $th_krm = 0.2$ and $th_rm = 0.2$;
- For *Test_3*, $th_sim = 0.8$, $th_krm = 0.3$ and $th_rm = 0.3$;
- For *Test_4*, $th_sim = 0.8$, $th_krm = 0.2$ and $th_rm = 0.2$;

³ We used *Open Refine* (<http://openrefine.org>) a powerful tool to work with messy data.

Table 1. Evaluation of experimental results.

Precision (%) for data sets				
Query	Test_1	Test_2	Test_3	Test_4
q_1	8.33	0.71	8.33	0.71
q_2	100.00	100.00	100.00	100.00
q_3	100.00	7.69	100.00	7.69
q_4	100.00	100.00	100.00	10.00
q_5	100.00	3.70	100.00	3.70
q_6	100.00	100.00	100.00	100.00
Precision (%) for items				
Query	Test_1	Test_2	Test_3	Test_4
q_1	1.57	0.09	0.33	0.01
q_2	100.00	100.00	100.00	100.00
q_3	100.00	100.00	100.00	100.00
q_4	100.00	100.00	100.00	100.00
q_5	100.00	5.15	100.00	5.15
q_6	100.00	100.00	100.00	100.00

Table 2. Comparison of T Test_1 with the baselines obtained by means of *Google CSE* and *Apache Solr*.

Recall (%) for data sets			
Query	Google CSE	Apache Solr	Test_1
q_1	100.00	20.00	100.00
q_2	0.00	0.00	100.00
q_3	100.00	5.00	100.00
q_4	100.00	15.00	100.00
q_5	0.00	0.00	100.00
q_6	100.00	10.00	100.00
Precision (%) for data sets			
Query	Google CSE	Apache Solr	Test_1
q_1	20.00	6.67	8.33
q_2	0.00	0.00	100.00
q_3	5.00	2.63	100.00
q_4	15.00	2.56	100.00
q_5	0.00	0.00	100.00
q_6	10.00	100.00	100.00

In the first section of Table 1, we report precision as far as the retrieval of data sets is concerned, before filtering items. We do not report recall, because we obtained always (for the 6 queries of the experiment) a value of 100%: this means that the desired data sets are always retrieved; consequently, precision becomes the key metric to analyze. In this respect (i.e., as far as precision is concerned), we notice that in case of queries q_1 , q_3 and q_5 , the reduction of values for th_krm and th_rm (*Test_2* ad *Test_4*), affects the precision, meaning that the technique retrieves a larger number of data sets. Then, we expect that this phenomenon will negatively affect item precision, i.e., precision concerning retrieval of single items.

Observe that both recall and precision are insensitive to the variation of parameter th_sim .

Let us consider now the retrieval of single items. The second part of Table 1 reports the precision, while recall is not reported because, even in this case, we always obtained 100%. Again, as for data sets, all wished items are retrieved, so it is important to verify, through precision, how many non-wished items are provided in the results.

In this respect (i.e., as far as item precision is concerned), most of queries retrieve exactly the desired items and nothing more. This is not true for queries q_1 and q_5 . In particular, query q_1 retrieves a very large number of items that are not related to the original query: this is because the query is formulated in a very generic way, then rewriting the query with synonyms causes the search space to dramatically enlarge.

Looking at item precision for query q_5 , we notice that, for *Test_1* and *Test_3*, the thresholds th_krm and th_rm set to 0.3 determines a precision of 100%, while reducing them to 0.2 causes the search space to dramatically enlarge, obtaining a very poor item precision (meaning that a large number of false positive items are provided).

From our experiment, we can conclude that our technique behaves better with *Test_1* configuration, i.e., with $th_sim = 0.9$, $th_krm = 0.3$ and $th_rm = 0.3$. The reason is that our approach is greedy during the *VSM Data Set Retrieval*; then, in the *Schema Fitting* step the search space is narrowed and data set instances are downloaded. With $th_krm = 0.2$, the VSM retrieval becomes too greedy and retrieves data sets that are too far away from the rewritten query.

Finally, the insensitiveness to th_sim is determined by the $Max_N = 10$ parameter, i.e., the maximum number of queries to process, that keeps only the 10 neighbour queries with terms that are more similar to chosen alternative terms (synonyms).

Baselines. To complete the evaluation, we had to choose a suitable baseline. In effect, we identified two possible tools that could provide a suitable baseline: *Google CSE* and *Apache Solr*.

*Google Custom Search Engine (CSE)*⁴ is a version of *Google* search engine targeted to a specific web site. We exploited it as baseline, having targeted it to

⁴ <https://cse.google.com/>.

the *New York City Open Data Portal*, submitting query strings obtained composing terms from within our queries (obviously, without selection condition). We considered data sets provided in the first two pages.

Apache Solr [8], is an open source enterprise search engine, developed on top of *Apache Lucene*. Although it is necessary to preliminarily download and index the corpus; however, it is considered one of the best general purpose information retrieval tool currently available.

Table 2 shows the results of our comparison: the upper part compares recall, while the lower part compares precision.

As far as recall is concerned, notice that *Google CSE* works better than *Apache Solr*, providing 100% of recall when it succeeds; surprisingly, it shows an *on/off* behavior, showing a recall of 0% for queries q_2 and qw_5 . In contrast, our technique is able to retrieve all data sets, because it combines synonyms and lexicographic similarity.

As far as data set precision is concerned, obviously we obtain a precision of 0% for queries q_2 and q_5 , because neither *Google CSE* nor *Apache Solr* retrieves the wished data sets. Again, *Google CSE* performs better than *Apache Solr*: our explanation of this phenomenon is that *Google CSE* effectively adopt language dictionaries and is able to deal with synonyms. The reader can see that our technique, when working with *Test_1* configuration, is generally better, apart from the case of query q_1 , for which *Google CSE* retrieves less false positive data sets.

9 Related Works

Querying Open Data portals, with specific techniques providing the ability of selecting specific data items, is a young research area and, at the best of our knowledge, we are in the pioneering phase. However, many researchers are reasoning about Open Data management.

In [9] Khosro et al., present the state of art in Linked Open Data (LOD), with issues and challenges. Moreover, the authors motivate this topic by exploiting the projects analyzed in the five major computer science areas (Intelligence, Multimedia, Sensors, File System and Library), and present the future trends and directions in LOD.

The Open Data world is related to the Linked Data world. In fact, standardized proposals are typically used to describe published Linked Open Data. The RDF (*Resource Description Framework*) [10] is widely used for this purpose. Furthermore, the standard query language for RDF is called *SPARQL Standard Protocol and RDF Query Language* [11]. W.r.t. our proposal, it is very general and devoted to retrieve those data sets with certain features. However, highly skilled people in computer science are able to use it. In contrast, our query technique is very easy for not skilled people and closer to the concept of query in information retrieval.

Similar considerations are done in [12], where the authors presents *DBPedia*: the RDF approach is not suitable for non expert users that need a flexible and simple query language.

In this paper we do not consider *Linked Open Data*: we query a corpus of Open Data Sets, in general not related to each other, thus, not linked at all.

The idea of extending our approach to a pool of federated Open Data Corpora is exciting. A pioneer work on this topic is [13], but they still rely on SPARQL as query language.

The heterogeneity of Open Data asks for the capability of NoSQL databases. In [14], the authors report their experience with *Elasticsearch* (distributed full-text search engine [15]), highlighting strengths and weaknesses.

Both *Elastic Search* [15] and *Apache Solr* [8] demonstrated to be very efficient information retrieval tools and, for this reason, they have become very popular. However, they provide basic support to retrieval and, as we discussed in Sect. 8, they do not deal with dictionaries and semantic aspects of search.

As far as the previous version of our technique is concerned [1], the enhanced version presented in this paper demonstrated to be much more effective: this is due to the addition of semantics, on the basis of the *WordNet* dictionary, that provides synonyms for query terms.

Our technique has been implemented within a prototype that we called *Hammer*. In [7] we discussed the challenge of implementing the *Hammer* prototype in an efficient way, based on state-of-the-art technologies for Big Data. Notice that the heterogeneity of Open Data asks for such modern technical solutions: in fact, many data sets are published as JSON data sets. JSON has become the de-facto standard for exchanging information in the Big Data world and, besides retrieval, NoSQL databases are providing support for storing and processing large volumes of JSON data sets. Novel query languages and frameworks for specifying complex transformations on possibly geo-tagged JSON data sets are becoming necessary, as shown in [16,17].

10 Conclusion

This paper presents an evolution of the technique we presented in [1] to retrieve items (rows in CSV files or objects in JSON vectors) contained in data sets published by an open data portal. The main characteristics of the basic approach are kept: users blindly query the published corpus; the technique both focuses the search on relevant terms and expands the search by generating *neighbour queries*. The main enhancement introduced in the technique concerns the query rewriting mechanism: in the new version, we do not rely only on lexicographic similarity but also on semantic similarity; in other words, we replace terms with synonyms obtained by means of *WordNet* dictionary.

To validate the technique, we performed experiments on the *New York City Open Data Portal*. A pool of 6 queries allowed us to prove the goodness of our approach, that significantly improved performance w.r.t. the previous version of the technique. Furthermore, in order to better evaluate the effectiveness, we compared the results with those provided by two distinct baselines, i.e., *Google Custom Search Engine (CSE)* and *Apache Solr*. We observed that our technique performs better both in terms of recall and precision.

Currently, the technique must be further improved as far as the precision is concerned: in fact, we are going to study how to reduce the number of retrieved false positive data sets and, consequently, the number of false positive items, that remain the weak point of the technique when the query is formulated in a too generic way.

Certainly, one direction we are going to follow to improve the effectiveness of the technique is the exploitation of explicit correlations between terms, that can be obtained by performing a preliminary analysis of correlated terms in the meta-data catalog.

Nevertheless, execution times are also a relevant issue in this domain, that is related with the Big Data world. In fact, we want to go beyond what we did in [7], where we addressed this issue applying modern technologies for Big Data processing.

References

1. Pelucchi, M., Psaila, G., Toccu, M.: Building a query engine for a corpus of open data. In: Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST-2017) INSTICC, pp. 126–136. ScitePress, Porto (2017)
2. Manning, C.D., Raghavan, P., Schütze, H., et al.: Introduction to Information Retrieval, vol. 1. Cambridge University Press, Cambridge (2008)
3. Miller, G.A.: Wordnet: a lexical database for English. *Commun. ACM* **38**, 39–41 (1995)
4. Winkler, W.E.: The state of record linkage and current research problems. In: Statistical Research Division, US Census Bureau, Citeseer (1999)
5. Jaro, M.A.: Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *J. Am. Statis. Assoc.* **84**, 414–420 (1989)
6. Liu, J., Dong, X., Halevy, A.Y.: Answering structured queries on unstructured data. In: WebDB, vol. 6, Citeseer, Chicago, Illinois, USA, pp. 25–30 (2006)
7. Pelucchi, M., Psaila, G., Toccu, M.: The challenge of using map-reduce to query open data. In: Proceedings of the 6th International Conference on Data Science Technologies and Applications DATA-2017, INSTICC. ScitePress, Madrid (2017)
8. Shahi, D.: Apache solr: An introduction. In: Apache Solr. Springer, Heidelberg, pp. 1–9 (2015). https://doi.org/10.1007/978-1-4842-1070-3_1
9. Khosro, S.C., Jabeen, F., Mashwani, S., Alam, I.: Linked open data: towards the realization of semantic web - a review. *Indian J. Sci. Technol.* **7**, 745–764 (2014)
10. Miller, E.: An introduction to the resource description framework. *Bull. Am. Soc. Inf. Sci. Technol.* **25**, 15–19 (1998)
11. Clark, K.G., Feigenbaum, L., Torres, E.: Sparql protocol for rdf. World Wide Web Consortium (W3C) Recommendation, p. 86 (2008)
12. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: a nucleus for a web of open data. In: Aberer, K., et al. (eds.) ASWC/ISWC - 2007. LNCS, vol. 4825, pp. 722–735. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76298-0_52
13. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: Fedx: a federation layer for distributed query processing on linked open data. In: The Semantic Web: Research and Applications. Extended Semantic Web Conference, vol. 6644, pp. 481–486. Springer, Heidelberg (2011). <https://doi.org/10.1007/978-3-642-21064-8>

14. Kononenko, O., Baysal, O., Holmes, R., Godfrey, M.: Mining modern repositories with elastic search. In: MSR, Hyderabad, India, 29–30 June 2014
15. Gormley, C., Tong, Z.: *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine*. O'Reilly Media, Inc., Massachusetts (2015)
16. Bordogna, G., Capelli, S., Psaila, G.: A big geo data query framework to correlate open data with social network geotagged posts. In: Bregt, A., Sarjakoski, T., van Lammeren, R., Rip, F. (eds.) *GIScience 2017. LNGC*, pp. 185–203. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56759-4_11
17. Bordogna, G., Ciriello, D.E., Psaila, G.: A flexible framework to cross-analyze heterogeneous multi-source geo-referenced information: the J-CO-QL proposal and its implementation. In: *Proceedings of the International Conference on Web Intelligence*, pp. 499–508. ACM (2017)



A Taxonomy for App-Enabled Devices: Mastering the Mobile Device Jungle

Christoph Rieger¹(✉) and Tim A. Majchrzak²

¹ ERCIS, University of Münster, Münster, Germany
christoph.rieger@ercis.de

² ERCIS, University of Agder, Kristiansand, Norway
tima@ercis.de

Abstract. While the term application is known for a long time, what we now refer to as mobile apps has facilitated task-oriented, interoperable software. The term was initially only used for smartphones and tablets, but desktop software now is also referred to as apps. More important than the wording, however, is the trend towards app-enablement of many further kinds of devices such as smart TVs and wearables. App-enabled devices usually share some characteristics and developing apps is often similar. However, many complexities must be mastered: Device fragmentation and cross-platform app development already are challenging when only considering smartphones. When trying to grasp the field as a whole, app-enabled devices appear as a jungle: it becomes increasingly hard to get an overview. Devices might not be easy to categorize let alone to compare. Investigating similarities and differences is not straightforward, as the outer appearance might be deceiving, and technological peculiarities are often complex in nature. This article aims at mastering the jungle. For this purpose, we propose a taxonomy for app-enabled devices. It provides clear terms and facilitates precision when discussing devices. Besides presenting the taxonomy and the rationale behind it, this article invites for discussion.

Keywords: App · Mobile app · Taxonomy · Categorization
Smart devices · Wearable · Smartphone · Tablet

1 Introduction

The continuous growth of the mobile device market [2] and the recent emergence of devices such as smart watches [3] and connected vehicles [4] has attracted much attention from academia and industry. In the past decade, particularly

This article greatly extends the short paper [1] presented at WEBIST 2017. It has been updated to reflect the latest developments, includes new content based on additional work as well as on the discussions at the conference, and has been amended with a comprehensive discussion. Please note that verbatim content from the short paper is not explicitly highlighted but for figures and tables already included there.

the app ecosystem facilitated a trend towards task-oriented, interoperable software, arguably started with the advent of Apple’s iPhone in 2007 [5] and the App Store in 2008 [6]. For *traditional* mobile devices (i.e. smartphones and tablets), the competition has yielded two major platforms (Android and iOS). However, whether these two will prevail can hardly be estimated, yet. Moreover, developing for such devices in a unified way is still not possible in all cases and with ease (cf. also [7]). Several approaches for cross-platform development have been proposed to avoid the costly re-development of the same app for different platforms (cf., e.g., [8,9]).

Technological development has continued in the meantime and many new device types have emerged. Most of them fall under the umbrella term *mobile devices* and are concerned by the research field of *mobile computing*. Typically, they are more-or-less *app-enabled*. While app enablement is no fixed or even defined term (to the best of our knowledge), it can be understood as follows:

An app-enabled device provides hardware that allows it to be used for multiple (typically many) purposes and in changing contexts – possibly even unforeseen by the device manufacturer – while the actual versatility of the device is achieved through means of extensible software that comes in small, interchangeable pieces which are usually provided by third parties.

Thus, it typically are apps that make such devices particularly useful and that extend the possibilities they offer. However, mobile devices that follow our rough definition differ greatly in intended use, capabilities, input possibilities, computational power, and versatility, to name just a few aspects. In early visions of a world connected by ubiquitous mobile devices, these were only thought of as tabs, pads, and boards [10]. So-called “smart devices” such as smart watches and smart TVs are most prominent in the realm of consumer devices and exhibit double-digit sales growths over the past years [11,12], but plenty of possibilities exist with regard to the physical embodiment of virtual assistants. Furthermore, hardware in professional contexts can be surprisingly similar to consumer-hardware; apps can make them seem even more akin. Lines towards sensor-driven devices for the Internet of Things (IoT) are often blurred and it is not always clear how to properly categorize a device [13]. This makes it hard to discuss, or, actually, to even correctly name them. The resulting blurriness makes it hard to delimit research and practical work. Much worse, when speaking and writing about mobile devices, the level of precision is often not as high as it is when well-known concepts are discussed. While this is normal for emerging fields, it is particularly noticeable for work on mobile computing. To our observation, there are only slow improvements.

We believe that more precision in speaking and writing will eventually also be beneficial for research on mobile devices and their app-enablement. These devices provide a plethora of new opportunities for intelligent and context-adaptive software. At the same time, they pose technical challenges regarding the development for new platforms and regarding heterogeneous hardware features. Interestingly, these challenges can be quite similar despite seemingly very different devices, as they can be completely different despite originating from the same

kind of device. Moreover, app-enablement does not necessarily bring compatibility and portability. Naturally, running the same app on a variety of devices is normally desirable. If we still rely on cross-platform approaches and search for a development unifier merely for smartphones and tablets [7], developing for heterogeneous mobile devices is an endeavour far greater in complexity. It would probably be ideal to reach something like a progression in functionality: the same app would function on many devices but respectively provide the highest level of functionality achievable on the given hardware and with the available other software. It must be doubted, however, that such an ideal can be reached as long as we do not even properly know *what* we are talking about.

While a plethora of case studies and contributions for individual device types – mainly focused on smartphones and tablets – can be found in the scientific literature (e.g., [14–18]), a comprehensive study of the general field of app-enabled devices is missing. With our WEBIST position paper [1], we set out to close this gap by contributing a taxonomy for app-enabled consumer devices. This contribution got favourable comments, encouraging us to provide an extension of our work with this article. The taxonomy aims at

- helping authors to clearly express what kind of device(s) they refer to,
- providing researchers and practitioners with more discriminatory power when referring to topics from modern mobile computing, and
- giving the general public a more straightforward understanding of similarities and differences between devices, both technically and tangibly.

Similar as in the prior paper, we have put much effort into literature work (cf. the next section), although the useful literature remains scarce. While the taxonomy has only been slightly updated to reflect the latest developments, we delve deeper into the theoretic dimension and also extend our discussion. Therefore, the work presented in this paper keeps a research-in-progress flavour, since it is impossible to suggest that our taxonomy is in its final state. However, it should be considered sufficiently stable for practical application. Any follow-up work from now on will honour this by either providing downward compatibility and (or alternatively) by explicating changes. We believe that more work will continue to be required; while we of course hope for this article to become a state-of-the-reference, it should also stimulate further work. The mid-term goal remains to be a *de-facto* standard.

The remainder of this article is structured as follows. Section 2 takes an updated look at relevant literature in the narrow sense; related work to specific aspects is referenced throughout the paper. Then, our proposal for a taxonomy is presented in Sect. 3. Section 4 discusses the taxonomy with regard to its current and future applicability. Finally, we conclude and give an outlook in Sect. 5.

2 Related Work

If you consider the topic of our article broadly, a plethora of related work exists. Looking at it in more detail, hardly any closely-related approaches can be cited.

This is not really surprising: all papers that deal with apps and app-enabled devices must (at least implicitly) explain *what* they actually deal with. However, no systematic work exists that defines kinds of devices, modes of app-enablement, notions of mobility of devices, and so on.

Particularly since Apple’s iPhone founded the *smartphone* device class, which soon saw many devices follow, many papers have been published on the *modern* notion of mobile computing, centring around devices that are propelled by apps. However, even overview papers typically focus on *one* category of devices. For example, [19] classify apps by usage states but limit themselves to smartphones. Moreover, the scientific literature so far has only rudimentarily captured the latest developments in device development. [20], for instance, provide an overview of smart watch app markets with focus on the type of apps as well as privacy risks through third party trackers.

To make sure that we do not miss an existing taxonomy (or similar work), we conducted an extensive literature search. We focus on work from 2012 or later, where the first broader range of smart watches such as the Pebble had already been presented. Together with the increasing variety in devices, new operating systems have appeared since then. Examples are Android Wear and watchOS, which focus on wearable devices [21,22] as well as webOS and Tizen, which address a wider range of smart devices [23,24]. Additionally, also the app ecosystems have matured, with HTML5 gaining momentum and possible technological unifiers such as progressive web apps (PWAs) [7] emerging.

In our search, we deliberately excluded the keywords *application* and *system*. The first yielded many results that were not applicable since the term was mostly used to mean *utilization* of something. The latter had originally been used to describe e.g. cyber-physical systems but now proved to be too generic. Also, the *medical* area was excluded as these papers focus on apps for therapeutic purposes and do not contribute to the question of app-enabled devices. We thus used the following search string in the Scopus database:

```
TITLE-ABS-KEY(
  (app-enabled OR app OR app-based)
  AND
  (mobile OR smart OR intelligent OR portable)
  AND
  (device OR vehicle OR “cyber-physical system” OR CPS OR gadget)
  AND
  (classification OR categorization OR overview OR comparison OR review →
  OR survey OR framework OR model OR landscape OR “status quo” →
  OR taxonomy)
)
AND PUBYEAR AFT 2011
AND ( EXCLUDE ( SUBJAREA, “MEDI” ))
```

A search on 01-08-2017 yielded 1,268 results. Of these, not a single paper provided an approach for classification, let alone a complete taxonomy. To complicate matters, some papers mention that there are other *smart* devices than

smartphones and tablets but do not go into detail. Only four papers went beyond a perspective on “classical” mobile devices: Some authors focus on specific combinations of devices, including Neate et al. [25] who analyse the use case of second screening that combines smart TVs with additional mobile devices and Singh and Buford [26] who describe cross-device team communication apps for desktop, smartphones and wearables. Regarding more generalized approaches, Queirós et al. [27] focus on context-aware apps also suitable for novel mobile devices using the example of an automotive app. Finally, Koren and Klamma [28] considered the integration of heterogeneous Web of Things device types by adopting a middleware approach.

In summary, the result set reveals no closely related work to which we can limit ourselves. However, we can draw from a myriad of sources that tackle *some* aspects that are relevant for a taxonomy of app-enabled devices. This finding aligns with the motivation for our paper. Obviously, other authors struggled with putting different device categories into context because no proper framing exists.

Despite not necessarily focussing on multiple device categories, work on *cross-platform app development* is conceptually related. Usually, cross-platform development exclusively targets traditional mobile devices such as smartphones and tablets, e.g. “the diversity in smart-devices (i.e. smartphones and tablets) and in their hardware features; such as screen-resolution, processing power, etc.” [29]. However, considering the differences in platforms, versions, and also at least partly in the hardware is similar to considering a different type of device. In fact, the difference in screen size between some wearables (such as some smart watches) and smartphones with small screens is less profound than between the same smartphones and tablets. Therefore, comparisons that target cross-platform app development have paved the way towards this article. This particularly applies to such works that include an in-depth discussion of criteria, such as by [8, 30–32].

A part of the difficulty with related work is the term *app-enabled* (or *app-enablement*) by itself. While it is often said that devices are enabled by apps, or that apps facilitate their functionality, it is usually not explained *what* this exactly means. But in the simplest devices that make use of computer hardware, software plays an important role; in consequence, merely being capable of running software that fulfills more than basic functionality is not enough to describe the term.

The typical usage that we also follow is to denote an app-enabled device as one that by its hardware and foundational software (such as the operation system or *platform*) alone provides far less versatility than it is able to offer in combination with additional applications. Such apps are not (all) pre-installed and predominantly provided by third party developers unrelated to the hardware vendor or platform manufacturer; moreover, the possibilities provided by apps typically increase over time *after* a device has been introduced. In addition, apps may expose use cases not originally intended or even imagined. While this still is no profound definition, it provides a demarcation for the time being.

In particular, it rules out pure Internet-of-Things devices as well as computational equipment that only is occasionally firmware-updated or that is not built for regular interaction with human users.

3 Taxonomy of App-Enabled Devices

In the following, we describe the preconditions of a taxonomy before describing how a device categorization can be tailored. We deem three dimensions to be viable as the static structure for classification. Eventually, we propose a categorization that captures the status quo. It positions current and foreseeable future device classes according to this matrix.

3.1 Basic Considerations

Categorizing app-enabled devices is difficult: there is a wide variety of possible hardware features *across* all types of devices, which even further increases. For example, in the past fingerprint scanners were restricted to few notebooks but today also appear on smartphones because of simplified user authentication and changing security requirements with regard to the device purpose (consumer vs. commercial). If classes are set – such as the widely acknowledged distinction between smartphones and tablets – there still is a heterogeneity of device capabilities *within* each class. For instance, the first smartwatches offered only a few sensors. Current devices have many more sensors, and their characteristics can differ significantly depending on the target sector (such as low-end vs. high-end).

Any simple solution is prone to not sufficiently discriminate. For example, processing power does not differ a lot between smartphones and tablets anymore, and microphones are no distinguishing feature for voice-controlled devices. The fast-paced technological progress manifests as a constant stream of new devices, partly rendering previous devices obsolete. Moreover, device types converge, illustrated e.g. by the *phablet* phenomenon (devices that fall in between smartphones and tablets).

Mobility in the strict sense even is no exclusive feature; smart TVs for example are not really mobile. Cars with smart entertainment systems or even self-driving features might be app-enabled, but it can be disputed whether the whole car is the *device* and thereby the device is actually mobile by itself. As a result, a taxonomy of app-enabled devices mandates a more open categorization along several dimensions, allowing for partial overlaps and future additions. In the following, we present steps towards such a taxonomy.

3.2 Dimensions of the Taxonomy

We position app-enabled devices with regard to the three dimensions *media richness of inputs*, *media richness of outputs*, and the *degree of mobility*. Instead of enumerating concrete technologies that are available today or *may be* introduced

in the future, each dimension should rather be regarded as continuously increasing intensity and variability of the particular capability, with several exemplary cornerstones depicted in the following. This approach not only provides the highest degree of objectivity but also should keep the taxonomy flexible enough to capture future developments without actually changing the dimensions.

Media richness of inputs describes the characteristic user input interface for the respective device class. Thereby, it captures how human users can interact with a device. Additional machine-to-machine communication through the same or distinctive interfaces is *not* considered.

None refers to fully automated data input through sensors.¹

Pass-through represents the indirect manipulation through data exchange with an external device (which in turn might originate from user input) whose purpose is not solely to provide the user interface for the main device.²

Buttons including switches and dials, are (physically) located at the device and provide rather limited input capabilities.

Remote controls including also joysticks and gamepads, refer to dedicated devices that are tethered or wirelessly connected to the app-enabled device. Technically, they merely make use of buttons, switches, dials etc. but provide a richer experience due to being decoupled from the device.

Keyboards are also dedicated devices to control the target devices, but with more flexible input capabilities due to a variety of keys. Input still is discrete.

Pointing devices refer to all dedicated devices to freely navigate and manipulate the (mostly graphical) user interface, for example mouse, stylus, and graphic tablet. While these devices technically still provide discrete input, the perception of input is continuous.

Touch adds advanced input capabilities on the device itself, allowing for more complex interactions such as swipe and multi-touch gestures. Strictly speaking, within this category simple touch events and several forms of increasingly complex multi-touch gestures can be subdivided.

Voice-based devices are not bound to tangible input surfaces but can be controlled without haptic contact.

Gestures allow for a hands-free user interaction with the device, for example using gloves or motion sensing. Technologically, different solutions are possible, e.g. based on gyroscopes, cameras, and lidars [33, 34].

Neural interfaces can be expected to become the richest form of user inputs by directly tapping into the brain or nervous system of the human operator.³

As the second dimension, *media richness of outputs* describes the main output mechanisms for the respective device class. Similarly to the input, human users

¹ Strictly, most if not all input is done via sensors, but *none* at this point denotes no manual activity by a user.

² For example, an autonomous device with a companion smartphone app for remote handling can be subsumed under this category.

³ Since the possibilities of neural interfaces are yet very limited and any work so far is experimental, future developments *might* mandate splitting up this category into different kinds of neural interfaces.

are concerned as the receivers; possible machine-to-machine communication is not relevant for this dimension.

None refers to no user-oriented communication by the device itself. This applies to cyber-physical actuators with direct manipulation of real-world objects (e.g., switching on light).

Pass-through includes mechanisms that in general or in some situations do not produce human-directed output of their own but pass it through to a connected managing device (e.g., a smartphone) which retrieves information and handles user output.

Screen output is the prevalent form of user communication found in app-enabled devices. Although a clear subdivision is not possible, several classes are typically observed, ranging from tiny screen displays ($<3''$) to small screens such as for smartphones ($<6''$), medium screens for handheld devices ($<11''$), large screens ($\leq 20''$), and usually permanently installed huge screens $>20''$.

Projection refers to the first type of disembodied device output to a device-external surface without physical contact.

Voice-based output extends the disembodiment with auditive output to communicate with the user without physical contact.

Augmented reality includes virtual reality applications and hologram representations, further increases the richness of device outputs by modifying or fully replacing the perceived reality around the user.

Neural interfaces connect directly to the user in order to achieve a tightly coupled human-computer interaction.⁴

Finally, the combination of input and output characteristics ignores different application areas of the respective device class. For example, intelligent switches and drones for aerial photography can both be remotely controlled and have no direct output, but can hardly be grouped as being in the same device class. Whereas several studies deal with usage characteristic particular devices such as smartphones (e.g., [35]), to the best of our knowledge no closely related work exists on context-dependent device usage across different mobile devices. Therefore, the *degree of mobility* describes the usage characteristics as the third dimension on a high level. With regard to trends such as ubiquitous computing [36], this dimension also reflects the pervasiveness and integration of mobile devices in everyday activities – from on-demand usage of stationary devices to always connected autonomous assistants.

Stationary devices are permanently installed and have no mobile characteristics during use.

Moveable devices can be carried to the place of use. This includes an “on-the-go” utilization, such as a smartphone being used while walking.

⁴ Similarly to the considerations for the input, it will need to be seen whether neural interfaces for output require some form of subdivision. Technology so far is in an early experimental state.

Wearable devices are designed for a more extensive usage and availability through the physical contact with the user. In contrast to “mobile”, transporting the device is implicit and often hands-free.

Self-moving devices provide the capability to move themselves (directly or indirectly controlled by the user). Ultimately, autonomous devices represent the richest form of mobility for app-enabled devices.

3.3 Categorizing the Device Landscape

The proposed dimensions allow for an initial categorization of the device landscape. Figures 1, 2 and 3 (pages (9) to (11)) visualize the three-dimensional categorization of different device classes using three two-dimensional projections for better readability. Also, Table 1 summarizes the device classes discussed in this paper.

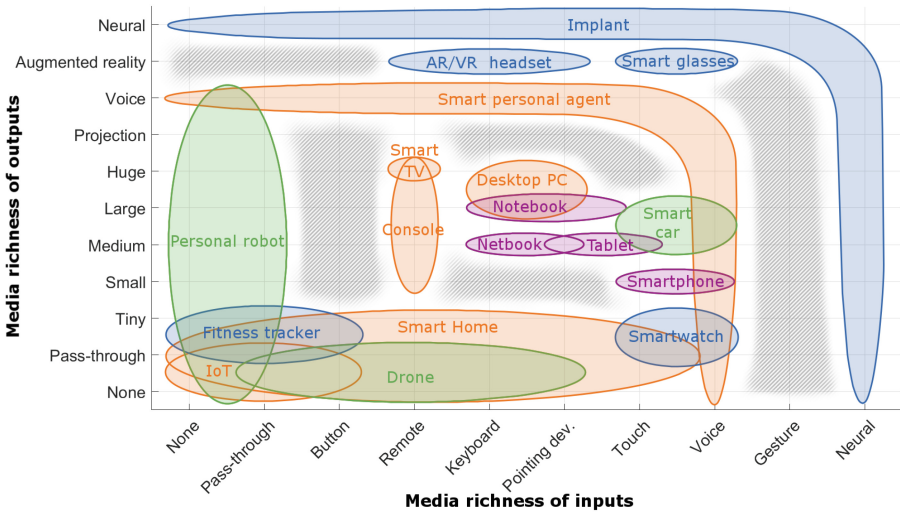


Fig. 1. Matrix of input and output dimensions (adapted from [1]).

As depicted in Fig. 1, many devices classes can be assigned to distinct positions in the two-dimensional space of input/output media richness. However, it should be noted that the ellipses represent (current) major interaction mechanisms within the device classes. For example, *smartphones* also have a few physical buttons but are mainly operated by touch input. Individual devices may also deviate from the presented position, for instance specialized or experimental devices that do not (yet?) constitute a distinct class of devices. Additionally, devices might be extended. For example, through special plugs computer mouses can typically be attached to smartphones. Since this is normally meant for debugging purposes, “pointing devices” would not normally be considered an

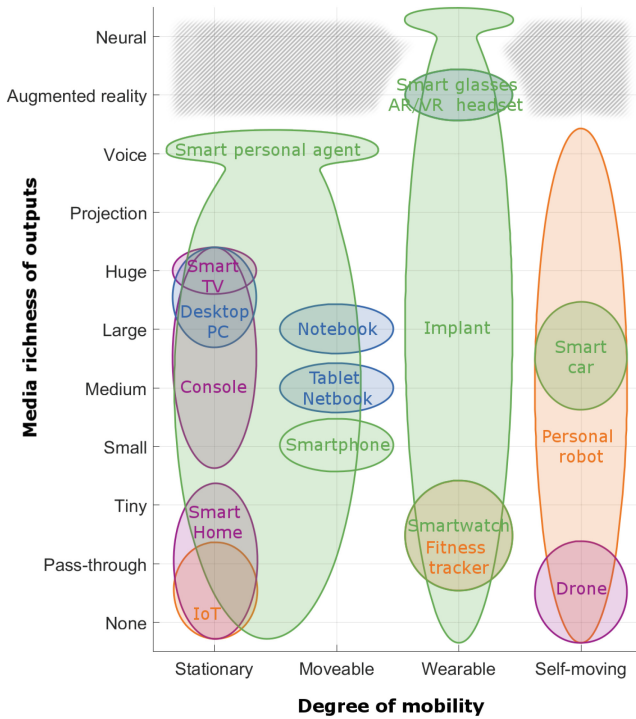


Fig. 2. Matrix of output and mobility dimensions (adapted from [1]).

input for smartphones. Similarly, so-called pico projectors allow image projection from smartphones and tablets. They are no typical mean for output, although this might change in the future.

Not all devices falling into a device class must necessarily implement all possibilities of that class. Therefore, ellipses are a well-suited representation as opposed to, e.g., the maximum value for the respective devices. A good approximation would be to consider at least 80% of all devices to match a category, with the lowest and the highest decile being outliers. For example, *convertibles* as hybrid devices between keyboard-based notebooks and touch-oriented tablets are not considered as they still represent a small minority in both categories.

The chosen level of abstraction implies that the taxonomy *dimensions* are intended to be rather static. Instead of chasing the actual technological development to reflect the latest emergence of devices, only seldom and slow changes are necessary to keep them up to date. Nevertheless, the categorization of *classes* is more dynamic and will need to be regularly checked for continued relevance. Moreover, classes might need to be split or at least be adapted regarding their placement on the dimensions' continuum when new possibilities arise. Thus, we explain some noteworthy classes exemplarily and rely on the general understanding of the well-known classes (such as smartphones).

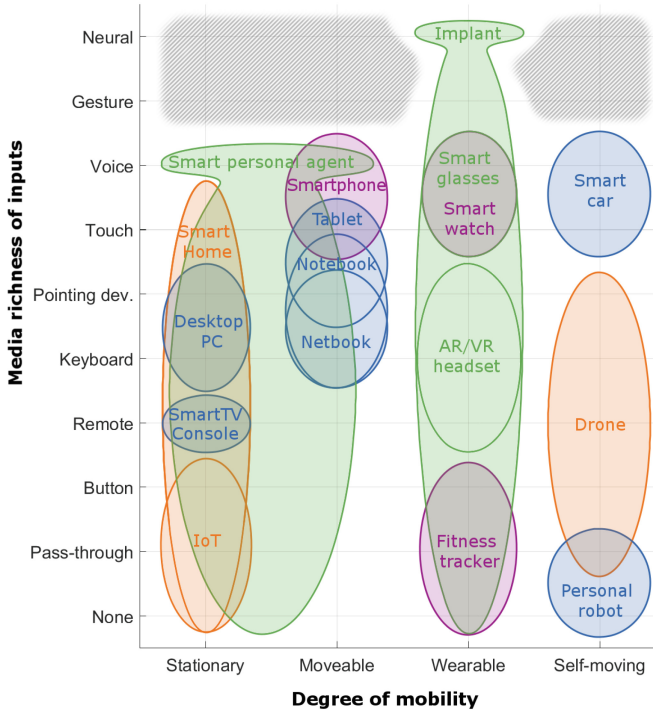


Fig. 3. Matrix of input and mobility dimensions (adapted from [1]).

Figure 1 reveals differences in the specificity (i.e., represented size) of the device classes. Some of them fill specific spots in the diagram, either due to technical restrictions (*smart TVs* evolved from traditional remote-controlled TVs with large screens) or special purposes (*smart glasses* enable hands-free interaction and visualization). Less specific device classes exist for two reasons. On the one hand, terms such as *smart home* comprise every technology that relates to a specific domain, subsuming very heterogeneous devices – thereby such a class represents an excellent high-level overview yet a poor low-level discriminating power. On the other hand, *underspecified* device classes such as *implants* and *smart personal agents* are presented *as they are* due to their novelty; there are few devices on the market and a high level of uncertainty must be ascertained regarding future hardware characteristics and interaction patterns.

Differences in the device classes can also be explained with regard to *media richness theory* (MRT). MRT describes a corridor of effective communication with matching levels of message ambiguity and media richness [37]. When applying this idea to the input and output characteristics of app-enabled devices, similar observations can be made. For example, *IoT* devices have only rudimentary possibilities for direct user input but also give not much feedback in return. Notebooks allow for medium levels of input richness through keyboard

Table 1. App-enabled device classes and their position in the continuum.

Device class	Input richness	Output richness	Degree of mobility
AR/VR headset	Remote - Pointing	Augmented reality	Wearable
Console	Remote	Small - Huge	Stationary
Desktop PC	Keyboard - Pointing	Large - Huge	Stationary
Drone	Pass-through - Pointing	None - Pass-through	Self-moving
Fitness trackers	None - Button	Pass-through - Tiny	Wearable
Implant	Neural	<i>any</i>	Wearable
	<i>any</i>	Neural	Wearable
IoT	None - Button	None - Pass-through	Stationary
Netbook	Keyboard - Pointing	Medium	Moveable
Notebook	Keyboard - Touch	Large	Moveable
Personal robot	None - Pass-through	None - Voice	Self-moving
Smart car	Touch - Voice	Medium - Large	Self-moving
Smart glasses	Touch - Voice	Augmented reality	Wearable
Smart home	None - Touch	None - Tiny	Stationary
Smart personal agent	Voice	None - Voice	Stationary-Moveable
	None - Voice	Voice	Stationary-Moveable
Smartphone	Touch - Voice	Small	Moveable
Smart TV	Remote	Huge	Stationary
Smartwatch	Touch - Voice	Pass-through - Tiny	Wearable
Tablet	Pointing - Touch	Medium	Moveable

and mouse input, with large screens as more flexible output capabilities. Furthermore, *smart glasses* directly embed their output into the real world by projection. Consequently, their voice-based input is equally rich in order to handle complex user interactions.

Figure 2 depicts the combination of output media richness and mobility. Unsurprisingly, a general tendency towards large screen output for stationary devices can be observed. With increasing mobility, output capabilities develop in two directions. On the one hand, screen sizes tend to diminish, from small screens on *smartphones* to very limited *fitness tracker* screens and screen-less *drones*. On the other hand, output capabilities become richer and overcome traditional screen-based approaches due to recent technological developments enabling intangible outputs, for instance *augmented/virtual reality (AR/VR) headsets*. It can also be observed that device classes with a high degree of

mobility are more variable and occupy larger spaces of the continuum. This is potentially caused by a fragmentation into various domains of application, or their novelty of appearance with insufficient time to establish wide-spread interaction patterns. Especially autonomously moving devices such as *smart cars* and *personal robots*, are driven by the increased availability of sensor technology and not restricted to particular output capabilities.

Finally, Fig. 3 visualizes the relationship between input media richness and mobility. Usually, an increasing degree of mobility entails less physical input mechanisms with dedicated buttons and keys. This might be attributed to practicability reasons, for example using voice commands is easier for wearable *smart glasses* than requiring dedicated input devices. In addition, smarter devices are usually more complex with regard to their output, and equally sophisticated input capabilities are necessary to match this level as explained by media richness theory. *Consoles*, for instance, provide basic navigation functionalities. *Desktop personal computers* and *notebooks* can be equipped with intelligent software such that keyboard and mouse are helpful means for interaction, and *smart personal agents* integrate advanced interpretation mechanisms that allow for voice-based communication in everyday situations.

MRT also partly explains why there are areas in the continuum with no assigned device class. Rich forms of user input such as gestures overcomplicate interactions for devices that have just small screens and therefore are typically equipped with limited sensing and processing resources. On the other extreme, devices with barely a few buttons do not provide sufficiently flexible input capabilities to manipulate large screens (such as several fingers multi-touch on a small smart watch). Of course, empty spaces in the taxonomy might also be caused by a lack of technological progress or use cases so far. Thus, they might actually be filled by future devices, or existing classes might “stretch” into these areas. For example, voice interfaces just recently emerged as mainstream technology in various devices from smartphones to smart home applications but augmented reality devices are still an active field of research. In general, with the evolution and differentiation of input media, existing device classes might extend towards further areas or even converge. For example, consider convertibles, such as the Lenovo Yoga Book, which represent hybrid devices between keyboard-based *notebooks* and touch-optimized *tablets* utilizing docking or folding mechanisms. Also, the evolution of one device class might render another obsolete; this can currently be observed with smartwatches cannibalizing the market for fitness trackers with more advanced input and output capabilities.

4 Discussion

The field of modern mobile computing does not show signs of less rapid progress. It, thus, is likely that amendments will need to be made. Additionally, we will need to keep updating the taxonomy once it has been acknowledged by the scientific community. Moreover, a taxonomy should be appealing for the use by practitioners, particularly in a field where scientific research and technological

progress go hand-in-hand. Therefore, this section presents ideas for discussion that go beyond the narrower focus of Sect. 3.

4.1 Alternative Categorization Schemes

Devices can be categorized according to other device features. Not all are compatible with our taxonomy, nevertheless we deem several of them noteworthy.

Simple schemes such as a categorization by hardware feature (e.g., camera resolution, raw computing power, touch screen availability) or usage (e.g., business, entertainment, sports, or communication) fail to provide clear criteria for a taxonomy. While they may even pose discriminatory power, they do not necessarily help with forming adequate classes of devices. In particular, a fast adaptation and convergence of available technologies could be observed in the past years. Using simple hardware features for categorization would thus be prone to quickly becoming obsolete. To give some examples: so-called *phablets* blur the lines between smartphones and tablets; cameras with resolutions a few years ago only imaginable in professional photography equipment now are routinely built into many mobile devices; and gyroscope sensors have found wide-spread adoption in a variety much mobile hardware for a variety of purposes.

Matrix-based categorizations allow for a better juxtaposition on two dimensions, for instance regarding the input and output characteristics of app-enabled devices. However, the heterogeneity of devices within a device class provides insufficient discriminating power. For example, medium-sized, touch-based screens are usual interfaces both for tablets and for the infotainment systems within smart cars. Similarly, distinguishing between apps for embedded or stand-alone devices is not always possible due to different types of device integrations within a device category (cf. e.g. [4] for smart cars).

Therefore, the third dimension chosen for our taxonomy adds the degree of mobility to distinguish between similar device hardware in different usage contexts. Other potential approaches for categorizing devices include the degree of integration, automation, or intelligence attainable or provided by the device. This reaches from simple input/output devices with limited app interaction (such as fitness trackers), to interoperable software (such as smartphones), highly cross-linked and automated devices (in the IoT or smart home field), and finally to intelligent machines. While we deem it reasonable to discuss such an optional fourth dimension, we do not think the taxonomy would profoundly gain more discriminatory power. The added complexity would not be justified as the underlying assumption of increasing processing complexity is to some extent already encoded in the richness of inputs and outputs.

An alternative or possibly additional means for categorization is a *graph*, more specifically a *tree*. This way, categorization would get a hierarchical character that could for example honour development history and be subdivided. Additionally, this representation would be well suited to reveal similarities in particular features. If shown as a *polytree* such as sketched in Fig. 4, even complex dependencies could be displayed (a smart watch, for example as a combination of wrist-worn fitness trackers and basic smartphone functionality). However,

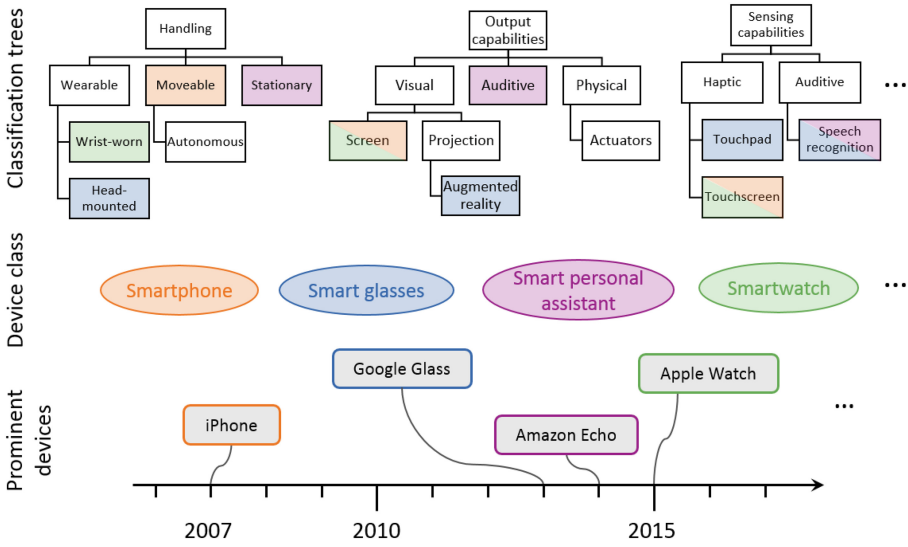


Fig. 4. Exemplary alternative classification approach.

while a tree representation surely is charming for its depiction of dependencies and historical connections, it poses far less discriminatory power than the taxonomy we designed. Blending both possibilities, for instance using a multi-layered representation of several trees that show the connection between device classes from different perspectives or for various criteria, would become too complex to be practical.

4.2 Further Development

Firstly, future discussion needs to include the demarcation of devices to be included. As argued earlier, mobility is not necessarily the proper boundary. App-enablement has proven to be feasible, yet we will need to find (or provide) a profound definition for it. This is an ongoing task.

Secondly, it needs to be determined how the taxonomy can be kept up to date. In many other cases, taxonomies have proven to be either too detailed and thus requiring constant adjustments, or too little detailed and thus lacking discriminatory power. In any case, taxonomies that are used in any not entirely static field ought to evolve.

Due to a restriction to three orthogonal dimensions and clearly distinguishable values in each of it, we are optimistic that the taxonomy will be future-proof. Nevertheless, proper ways of deciding when adaptations are needed and what developments can be reflected without changes need to be defined. As part of this, we will need to scrutinize how to handle the differences in precision regarding categories. For example, it is very well understood what a smartphone is;

smart homes, and to an even higher degree neural devices are (yet) diffuse with a lack of devices and applications to characterize them.

Thirdly, we so far have limited ourselves to consumer devices. This includes many devices that are *also* used for professional purposes, but arguably not all. Beyond that, some specialised devices are (so far) solely used for professional means. Examples can be found in industry, particularly in logistics. However, some of these might simply be subsumed by consumer devices. It could be said that, e.g., the devices used by parcel couriers are very similar to smartphones, despite the difference in form and the absence of a general purpose utilization. Moreover, commercial (and, similarly, military) devices might be derivatives of consumer hardware that has been “hardened” and more extensively tested. The same applies to special devices from areas such as healthcare or crisis prevention and response. While such devices typically have specific capabilities (such as error-tolerance), on an abstract level they again are very similar to general purpose hardware. Thus, an updated taxonomy could try to include non-consumer devices. However, due to the complexity that arises particularly with devices that are so specialised that information regarding them is scarce, we deem the current limitation justified. Additionally, if kinds of devices are seldom addressed in writing, including them in a taxonomy arguably is superfluous anyway.

Fourthly, it should be scrutinized how the taxonomy can be provided in a form that is useful for researchers *and* for practitioners. Most scientists know taxonomies for research topics enforced by publication outlets.⁵ Quite often these feel more like a “try to fit somewhere” game, particularly if a paper tackles a contemporary topic and the taxonomy provides little flexibility. If we want our taxonomy to be helpful for researchers, and – probably even harder to achieve – employed by practitioners, it needs to be easy to use yet powerful. Achieving this will be very valuable, as can e.g. be seen for cross-platform development, where new approaches can be clearly categorized by their characteristics. We think that our taxonomy should allow to put each device into exactly one class – choosing several applicable classes might be practical for the above named paper-theme categories, but we do not deem it practical for the purpose of our taxonomy.

The four discussion points have also illustrated the limitations of our work. Besides these issues that need to be worked on, an eventual verification of the taxonomy is mandated. Our planned work to further on this topic is sketched in the next section.

It would also be possible to develop the taxonomy towards an ontology (cf. [39]), possibly resulting in automated categorization aid which would also take ideas from the alternative polytree-based categorization as discussed in the previous section. For an unknown device, a decision tree could be traversed, leading to a prediction which kind of device is at hand. However, this would require rich semantic data (to allow inference), and it is currently not clear whether such an ontology would be considered to have much more value than the taxonomy already possesses.

⁵ An example is the ACM Computing Classification System, firstly presented in 1964 and revised in 1991, 1998, and 2012 [38] (cf. also [39]).

5 Conclusion and Outlook

With this article, we have proposed a taxonomy for app-enabled devices. It builds on a position paper presented in April 2017 – and it remains the first such work. The taxonomy is based on three dimensions: the media richness of inputs and of outputs, and the degree of mobility. Examined separately, each dimension is relatively simple. In combination, they provide high discriminatory power. This becomes particularly evident when categorizing the current device landscape. We have provided figures that support this assessment throughout this paper. In general, it has proven to be much easier to use “flat” representations of two dimensions at a time than to render a 3D model – at least for publication.

The presented taxonomy can be considered as a milestone, and we deem it to be static for now. Undoubtedly, progress in the field will mandate future changes, but these will rather lead to a new *version* of the taxonomy than to a *new* taxonomy. Nonetheless, this article should still act as an invitation for discussion. After all, the taxonomy is but a step towards a more unified view of mobile computing and a solidified theoretic base in this field. Moreover, future work will need to continue with keeping a systematic overview of app-enabled devices.

A better theoretic understanding of mobile computing, producible advice for practice, and word towards unified development mark the pillars of our future work. As a part of this, we will use the taxonomy and also put it up for further discussion, e.g. as part of conference talks. Additionally, we will now reach out to our partners from practice and ask them for an assessment. If the taxonomy will become well adopted, empirical work should follow.

As we already wrote in the position paper, we do not hope for our work to become “yet another computer science taxonomy”. Therefore, we hope that this article can illustrate the usefulness of a taxonomy and spark the interest for employing it.

References

1. Rieger, C., Majchrzak, T.A.: Conquering the mobile device jungle: towards a taxonomy for app-enabled devices. In: Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST), pp. 332–339. SciTePress (2017)
2. Statista Inc.: Global smartphone shipments forecast from 2010 to 2020 (2016). <https://www.statista.com/statistics/263441/>
3. Chuah, S.H.W., Rauschnabel, P.A., Krey, N., Nguyen, B., Ramayah, T., Lade, S.: Wearable technologies: the role of usefulness and visibility in smartwatch adoption. *Comput. Hum. Behav.* **65**, 276–284 (2016)
4. Coppola, R., Morisio, M.: Connected car: technologies, issues, future trends. *ACM Comput. Surv.* **49**(3), Article No. 46 (2016)
5. Apple Inc.: Apple reinvents the phone with iPhone (2007). <http://www.apple.com/pr/library/2007/01/09Apple-Reinvents-the-Phone-with-iPhone.html>

6. Apple Inc.: iPhone app store downloads top 10 million in first weekend (2008). <http://www.apple.com/pr/library/2008/07/14iPhone-App-Store-Downloads-Top-10-Million-in-First-Weekend.html>
7. Biørn-Hansen, A., Majchrzak, T.A., Grønli, T.M.: Progressive web apps: the possible web-native unifier for mobile development. In: Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST), pp. 344–351. SciTePress (2017)
8. Heitkötter, H., Hanschke, S., Majchrzak, T.A.: Evaluating cross-platform development approaches for mobile applications. In: Cordeiro, J., Krempels, K.-H. (eds.) WEBIST 2012. LNBP, vol. 140, pp. 120–138. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36608-6_8
9. El-Kassas, W.S., Abdullah, B.A., Yousef, A.H., Wahba, A.M.: Taxonomy of cross-platform mobile applications development approaches. *Ain Shams Eng. J.* **8**, 163–190 (2015)
10. Weiser, M.: The computer for the 21st century. *Sci. Am.* **265**, 94–104 (1991)
11. Statista Inc.: Global smart tv unit sales from 2014 to 2018 (in millions) (2017). <https://www.statista.com/statistics/540675/global-smart-tv-unit-sales/>
12. Statista Inc.: Smartwatches (2017). <https://www.statista.com/study/36038/smartwatches-statista-dossier/>
13. Ibarra-Esquer, J.E., González-Navarro, F.F., Flores-Rios, B.L., Burtseva, L., Astorga-Vargas, M.A.: Tracking the evolution of the internet of things concept across different application domains. *Sensors* **17**, 1379 (2017)
14. Heitkötter, H., Majchrzak, T.A., Kuchen, H.: Cross-platform model-driven development of mobile applications with MD2. In: Shin, S.Y., Maldonado, J.C. (eds.) Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC), SAC 2013, pp. 526–533. ACM (2013)
15. Jones, C., Jia, X.: The AXIOM model framework: transforming requirements to native code for cross-platform mobile applications. In: Ferreira Pires, L. (ed.) 2nd International Conference on Model-Driven Engineering and Software Development. IEEE (2014)
16. Chauhan, J., Mahanti, A., Kaafar, M.A.: Towards the era of wearable computing? In: Proceedings of 2014 CoNEXT on Student Workshop, CoNEXT Student Workshop 2014, pp. 24–25. ACM (2014)
17. Busold, C., Heuser, S., Rios, J., Sadeghi, A.-R., Asokan, N.: Smart and secure cross-device apps for the internet of advanced things. In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 272–290. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47854-7_17
18. Dageförde, J.C., Reischmann, T., Majchrzak, T.A., Ernsting, J.: Generating app product lines in a model-driven cross-platform development approach. In: 49th Hawaii International Conference on System Sciences (HICSS), pp. 5803–5812 (2016)
19. Jesdabodi, C., Maalej, W.: Understanding usage states on mobile devices. In: Proceedings of 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp 2015, pp. 1221–1225. ACM (2015)
20. Chauhan, J., Seneviratne, S., Kaafar, M.A., Mahanti, A., Seneviratne, A.: Characterization of early smartwatch apps. In: 2016 IEEE International Conference on PerCom Workshops 2016 (2016)
21. Google Inc.: Android wear (2016). <https://android.com/wear>
22. Apple Inc.: WatchOS (2016). <https://www.apple.com/watchos/>
23. LG Electronics: WebOS for LG Smart TVs (2016). <http://www.lg.com/uk/smarttv/webos>

24. The Linux Foundation: Tizen (2016). <https://www.tizen.org>
25. Neate, T., Jones, M., Evans, M.: Cross-device media: a review of second screening and multi-device television. *Pers. Ubiquit. Comput.* **21**, 391–405 (2017)
26. Singh, K., Buford, J.: Developing WebRTC-based team apps with a cross-platform mobile framework. In: 13th IEEE Annual Consumer Communications and Networking Conference, CCNC 2016 (2016)
27. Queirós, R., Portela, F., Machado, J.: Magni - a framework for developing context-aware mobile applications. In: Rocha, Á., Correia, A.M., Adeli, H., Reis, L.P., Costanzo, S. (eds.) *WorldCIST 2017. AISC*, vol. 571, pp. 417–426. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56541-5_43
28. Koren, I., Klamma, R.: The direwolf inside you: end user development for heterogeneous web of things appliances. In: Bozzon, A., Cudre-Maroux, P., Pautasso, C. (eds.) *ICWE 2016. LNCS*, vol. 9671, pp. 484–491. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-38791-8_35
29. Humayoun, S.R., Ehrhart, S., Ebert, A.: Developing mobile apps using cross-platform frameworks: a case study. In: Kurosu, M. (ed.) *HCI 2013. LNCS*, vol. 8004, pp. 371–380. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39232-0_41
30. Sommer, A., Krusche, S.: Evaluation of cross-platform frameworks for mobile applications. *Lecture Notes in Informatics (LNI) P-215*, pp. 363–376 (2013)
31. Dalmasso, I., Datta, S.K., Bonnet, C., Nikaein, N.: Survey, comparison and evaluation of cross platform mobile application development tools. In: 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC) (2013)
32. Rieger, C., Majchrzak, T.A.: Weighted evaluation framework for cross-platform app development approaches. In: Wrycza, S. (ed.) *SIGSAND/PLAIS 2016. LNBIP*, vol. 264, pp. 18–39. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46642-2_2
33. Lahiani, H., Kherallah, M., Neji, M.: Vision based hand gesture recognition for mobile devices: a review. In: Abraham, A., Haqiq, A., Alimi, A.M., Mezzour, G., Rokbani, N., Muda, A.K. (eds.) *HIS 2016. AISC*, vol. 552, pp. 308–318. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52941-7_31
34. Bhowmik, A.K.: 39.2: Invited paper: natural and intuitive user interfaces: technologies and applications. In: *SID Symposium Digest of Technical Papers*, vol. 44, pp. 544–546 (2013)
35. Hintze, D., Hintze, P., Findling, R.D., Mayrhofer, R.: A large-scale, long-term analysis of mobile device usage characteristics. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, pp. 1–21 (2017)
36. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of Things (IoT): a vision, architectural elements, and future directions. *Future Gener. Comput. Syst.* **29**, 1645–1660 (2013)
37. Daft, R.L., Lengel, R.H., Trevino, L.K.: Message equivocality, media selection, and manager performance: implications for information systems. *MIS Q.* **11**, 355 (1987)
38. *ACM Computing Classification System ToC* (2015). <http://www.acm.org/about/class>
39. Cassel, L.N., Palivela, S., Marepalli, S., Padyala, A., Deep, R., Terala, S.: The new ACM CCS and a computing ontology. In: *Proceedings of 13th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL)*, pp. 427–428. ACM (2013)



Attaining Role-Based, Mandatory, and Discretionary Access Control for Services by Intercepting API Calls in Mobile Systems

Yaira K. Rivera Sánchez^{1(✉)}, Steven A. Demurjian¹,
and Lukas Gnirke²

¹ University of Connecticut, Storrs, CT 06269, USA
ykrs@engineer.uconn.edu, Steven.Demurjian@uconn.edu

² Oberlin College, Oberlin, OH 44074, USA

Abstract. Mobile applications are quickly replacing traditional desktop computing for gaming, social media, email, web browsing, health and fitness, business usage, etc. Many of these mobile apps require that sensitive information (protected health information (PHI) and personally identifiable information (PII)) be displayed, accessed, modified, and stored. In the healthcare domain, there is a need for health information exchange (HIE) among patients and medical providers across a wide range of health information technology (HIT) systems such as electronic health records, e-prescribing, etc., all of which involve highly-sensitive data (PII and PHI) that is exchanged back and forth between the mobile application and its server-side repository/database. In the U. S. in 2015, the Office of the National Coordinator issued a report on certification rules for EHRs that has required that HIT vendors develop RESTful APIs for EHRs and other systems so that patients and medical providers using mobile health (mHealth) applications via the cloud can easily access their healthcare data from multiple sources. This necessitates the consideration that access control mechanisms are candidates to protect highly-sensitive data of such applications via the control of who can call which service. The paper presents the attainment of role-based (RBAC), mandatory (MAC), and discretionary (DAC) access control for RESTful API and cloud services via an Intercepting API Calls approach that is able to define and enforce users of mobile apps to limit the API/cloud services that can be invoked depending on a user's permissions. The presented Intercepting API Calls approach is demonstrated via an existing mHealth application.

Keywords: Access control · Application Programming Interface (API) Authorization · Mobile application Representational state transfer (REST) services Role-based access control (RBAC) · Mandatory access control (MAC) Discretionary access control (DAC)

1 Introduction

Mobile applications span a broad spectrum of complexity, including games, social networking, email, web browsing, financial management, health and fitness, etc. For both personal and business usage, there is a need to insure that access to secure information is controlled, ranging from protected health information (PHI) and personally identifiable information (PII) to confidential data that is displayed, accessed, modified, and stored. In healthcare, there has been a transition from paper-based to electronic health records (EHRs) with eight out of ten physicians in the U.S. utilizing EHRs in their practices [1]. Despite this progress, there is still a need for a significant next step to allow patients and medical providers to easily access healthcare data that is distributed across multiple EHRs and other health information technology (HIT) systems. To support these actions, health information exchange (HIE) for the interoperability across sources has the potential to reduce healthcare data expenses where healthcare institutions could save up to \$77.8B in the U.S. [2]. In support of the emergence of cloud computing in healthcare, the Meaningful Use Stage 3 [3] guidelines require all health information technology (HIT) systems (e.g., electronic health records (EHR), personal health records (PHR), etc.) to have API services to access, modify, and exchange health-related data. If services are the primary means of access, there must be a way to control who can invoke which service at which time. Access control mechanisms are commonly utilized to secure highly sensitive data in order to control which information each user can access/store in a particular system, with the proviso that disclosing the wrong information could lead to serious consequences [4]. The three dominant access control models [5] to achieve this are: role-based access control (RBAC) which defines roles with permissions on objects that are assigned to users; discretionary access control (DAC) where security policies are established based on the user's identity and authorization and can be delegated; and, mandatory access control (MAC) where sensitivity levels (Top Secret, Secret, Confidential, and Unclassified) are assigned to objects (classifications) and users (clearances) to control who can see what. This paper explores the usage of role-based (RBAC) [6, 7], mandatory (MAC) [8], and discretionary (DAC) [9] access control to manage which user of a mobile application is allowed to invoke specific API services. This required the evolution of RBAC and MAC to support permissions on services (as opposed to the usual object view) at a model level applied to a setting where a mobile application is using RESTful APIs and, by adding delegation permissions to services in support of DAC.

The work presented in this paper focuses on securing highly sensitive information (PII and PHI) that is present in many mobile applications and is accessible via an API, where the data transactions between a mobile app and a server are performed via invocations to the services of the API. This is achieved via the utilization of RBAC, MAC, and DAC in a two-phase process of definition and enforcement on a user of a mobile app with a given role to control which services of the app's API can be invoked. First, RBAC/MAC permissions are defined on a role-by-role/user basis, respectively, on the API services of the mobile app to identify which services can be called by which user by role utilizing the mobile app. Second, RBAC/MAC enforcement is achieved at runtime by intercepting each of the mobile app API calls of a user by role/user basis, respectively, in order to perform real-time RBAC/MAC permission checks. To supplement this two phase

process, delegation can be utilized to pass on RBAC and/or MAC permissions on APIs and their services. Our choice of RBAC and MAC to control APIs and their services is motivated by its wide usage for securing highly sensitive data for: corporate data [10] and, in EHRs [11]. Another motivation for the latter case is the emergent Fast Healthcare Interoperability Resources (FHIR) [12] standard for exchangeable healthcare resources that are accessible via an API from EHRs and other health information technology systems.

Through the mobile app API, we seek to provide a means for a user playing a role/clearance to be constrained to deliver/store data by limiting access to API services when utilizing the mobile app via the interception of the API calls, the actual invocation of services. According to [13], every API service should be verified to ensure that the user accessing the mobile app has the necessary permissions to manage the requested data. The Intercepting API Calls approach presented in this paper supports the interception of API calls by generating a new API at the service level that mirrors the original mobile app API (in terms of signatures) and serves as a wrapper which includes invocations to the original mobile app API to proceed based on RBAC/MAC/DAC checks that control the data that is displayed (delivered) and managed (stored). The larger intent of our research would be to define RBAC, MAC, and DAC permissions on API services and intercept calls for access control permission checks that determine the filtered information returned to the mobile app and control information that can be stored in the mobile application's server. To place the work presented in this paper in its proper perspective, we briefly highlight other published efforts. The work in this paper builds off of our previous work on authentication requirements for mobile apps [14] that was expanded to define an approach for role-based access control (RBAC) for mobile computing [15]. The work in this paper significantly extends [16] which focused only on RBAC and did not include MAC and DAC. The work in this paper operates on the direct API of the mobile application. A related effort [17] incorporates RBAC at the cloud side, to securely control via RBAC the cloud services that the mobile application services could invoke.

The remainder of the paper has eight sections. Section 2 states background concepts, motivates the Intercepting API Calls approach by explaining the important role of accessing information (especially PPI and PHI), and describes the Connecticut Concussion Tracker (CT²) mHealth app for tracking concussions from kindergarten to 12th grade. Section 3 presents the high-level processing of the classic architecture of the User Layer, Presentation Layer, Business Layer, and Data Layer and expands by including a discussion of the integration and usage of RBAC, MAC, and DAC. Section 4 presents a model for service-based RBAC, MAC, and DAC and, explores the underlying processing of the approach by examining the way that API services are categorized for RBAC and MAC. Section 5 examines the interactions and infrastructure for the combined RBAC/MAC/DAC approach for Intercepting API Calls. Section 6 explores the automatic generation of code for the RBAC, MAC, and DAC Intercepting API Calls enforcement via an algorithm. Section 7 illustrates the Intercepting API Calls approach for RBAC, MAC, and DAC utilizing the CT² mHealth application. Section 8 discusses related work in security and access control mechanisms for mobile applications, comparing and contrasting these efforts to our Intercepting API Calls approach. Finally, Sect. 9 concludes the paper.

2 Background Concepts, Motivation, and the CT² mHealth App

This section provides: background on role-based access control (RBAC) [6, 7], mandatory access control (MAC) [8], discretionary access control (DAC) [9], and APIs; motivation on the increasing role of APIs and a need for security; and, a review of the Connecticut Concussion Tracker (CT²) mHealth application. First, access control mechanisms are utilized to manage which permissions should be granted or denied in regards to the resources of a system or application. Three of the most popular mechanisms are role-based access control (RBAC), mandatory access control (MAC), and discretionary access control (DAC). In RBAC, users are assigned roles and each role contains different permissions, which contain the policies of which operations and objects a user with a particular role can have access to. In MAC, a security administrator assigns sensitivity levels (Top Secret (TS), Secret (S), Confidential (C), and Unclassified (U)) to objects (classifications) and users (clearances) to control who can see what where $TS > S > C > U$. In DAC, security policies (e.g., read, write, execute permissions) are established based on a combination of the objects and on the user's identity and authorization and can be delegated. For the purposes of this paper, we apply RBAC, MAC, and DAC concepts at the API level of the mobile app in support of the Intercepting API Calls approach to define by user/role which services of the API can be called at which times and under which conditions that are then enforced when a service is invoked by a user/role combination.

Second, in order to do data transactions between a server/database and a mobile application, many developers utilize the Application Programming Interface (API) concept. This consists of different tools, protocols, and libraries used to interface data to an application [18]. Basically, the client sends a request through the means of a URL, the API receives the URL and interprets it, and then sends this to the data source. The data source will then execute the request and send back a response to the API. The API encodes the response in a human readable format (e.g., JSON, XML) and sends the response in this format to the client. Some advantages of APIs are: utilized in several applications as most of them are modular (e.g., Facebook Graph API [19]); useful in applications that contain dynamic data (data that changes in a frequent manner); facilitate the sharing of data or processes between two systems; and, are highly interoperable [20]. The concept of API originated with traditional desktop devices and is now being heavily utilized in mobile applications. The Intercepting API Calls approach is aimed towards APIs that are built under the REST architectural style [21] and that use HTTP as a transfer protocol [22].

In terms of motivating the ideas in the paper, we acknowledge one of the most recognized options to display (deliver) and manage (store) dynamic data in a mobile app is to utilize the concept of API. However, before attempting to implement an API, one must evaluate their security risks and their effective management [23]. For example, consider the recent security breaches in Snapchat and Instagram APIs. Snapchat, a mobile app that enables users to view and send self-destructive pictures and videos [24], had a data breach that affected 4.6 million users [25]. The company quickly posted a statement revealing that the vulnerability allowed individuals to compile a database that

contained usernames and phone numbers of users of the mobile app and, that this problem came from their private API. To address this issue, Snapchat is attempting to identify which third-party applications offered in the iTunes store and Google Play store are accessing their private API and any application that uses it is accessing Snapchat's information without their permission [26]. Instagram, a mobile app that allows users to take pictures and share them with family and friends [27], had a password breach in 2015 [28]. The breach allowed a third-party application to steal more than 500,000 usernames and passwords, and used the information to post spam on Instagram accounts without permission. To remedy this, Instagram is now reviewing all of the applications that utilize their API and is adding new usage policies [29]. Clearly both public and private APIs need to be continuously secured and monitored to prevent disclosure of restricted information from occurring. To address this issue, a number of companies have added security and associated management mechanisms to APIs.

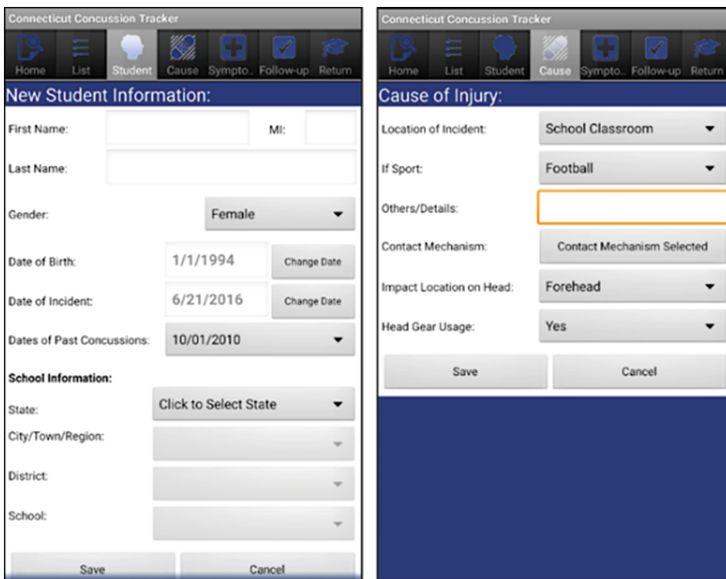


Fig. 1. Two screens of the CT² mHealth app [16].

Lastly, to serve as an example throughout the paper, the Connecticut Concussion Tracker (CT²) mHealth application, database, and its server are utilized. CT² is a collaboration between the Departments of Physiology and Neurobiology, and Computer Science & Engineering at the University of Connecticut and Schools of Nursing and Medicine in support of a new law passed in the state of Connecticut to track concussions of kids between ages 7 to age 19 in public schools (CT Law HB6722) [30]. The CT² application is for Android and iOS devices and utilizes a REST API in order to manage its data. The CT² mHealth app contains seven tabs ('Home', 'List', 'Student', 'Cause', 'Symptoms', 'Follow-up', and 'Return') where: the 'Home' tab allows the user to add a concussion, to retrieve an open case, or to find a student by name; the 'List' tab which contains the list of students the user has permission to view and, for each student gives

him/her the option to add a concussion or edit an existing one; the ‘Student’ tab (left screen in Fig. 1) allows the user to input the student’s general information (e.g., name, birthdate, school) and the date of concussion; the ‘Cause’ tab (right screen in Fig. 1) allows the user to specify how and where the concussion occurred; the ‘Symptoms’ tab allows users to record the symptoms the student had within 48 h and other pertinent data; the ‘Follow-up’ tab allows users to record the status of the student over time; and the ‘Return’ tab allows users to specify when the student can return to various activities at school.

3 High-Level Processing of the Intercepting API Calls Approach

This section explores the high-level processing of the Intercepting API Calls approach with an emphasis on the way that invocations from the mobile application to the mobile app API are intercepted. The Intercepting API Calls approach defines a new API (i.e., a new set of services) that mirrors the original mobile app API (in terms of the signatures) and serves as a wrapper and includes invocations to the original mobile app API to proceed based on access control checks that control the data that is displayed (delivered) and managed (stored). In this section, the high-level processing of the Intercepting API Calls approach is explored in detail; this approach offers the versatility of intercepting original API services (and their invocations), and by doing so, has no impact on the source code of the mobile application. We differentiate between three different APIs in the discussion: the *mobile app APIs* that are used by the mobile app (i.e., the original mobile app API services); the *intercepting mobile app API services* that has the same signatures as the mobile app original APIs to replace these and provide permission checks (i.e., mirrors the signatures of the original API services); and the *renamed mobile app API services* (i.e., the original mobile app APIs that are renamed to a set of corresponding services) that are wrapped by the intercepting mobile app API.

For the general architecture of a mobile app, we employ a client mobile app [31] augmented with the Intercepting API Calls approach. We focus on client applications since these are easier to maintain and assume that the app is always fully connected to the Internet. This assumes that all of the data is processed server-side and does not contain cache and local data. The architecture consists of four main layers as shown in the left side of Fig. 2: the User Layer which symbolizes the users of the mobile application; the Presentation Layer which consists of the UI components of the mobile application; the Business Layer which contains the logic of the mobile app (e.g., libraries, APIs, source code); and, the Data Layer which contains all of the data the mobile app manages (e.g., retrieves, inserts). The right side of Fig. 2 details the architecture of the Intercepting API Calls approach across the four layers in three groups. The first group, Role/Clearance/Delegation Assignment, involves the user layer and contains the users of the mobile app and their assigned roles/clearance/delegations. The second group, Define Access Control Permissions on API Services, spans the presentation and business layers and contains the original mobile app API services to retrieve/insert data from/into the data source. This group is utilized to define access control permissions on a role-by-role, clearance, and optional delegation basis that determine which original

mobile app API services are authorized to each role/clearance/delegation, which in turn is assigned to different users. Once access control permissions are defined on the mobile app API, our approach can intercept API services utilized by the mobile app in order to perform security and permissions checks. To transition from the second to third group, our Intercepting API Calls approach utilizes the data layer as a pass via the renamed API service invocations, and as a result, does not require modifying the source code of the mobile app in order to achieve. Lastly, the third group, Enforce Access Control Permissions on API Services, contains the RBAC, MAC, and DAC policies that need to be incorporated in the original data source(s) so that they can be enforced. This includes a new set of intercepting API services that must be defined and then utilized to replace the original mobile app API services to enforce the defined access control policies to control the data that is displayed (delivered) and managed (stored) on a user/role/clearance combination.

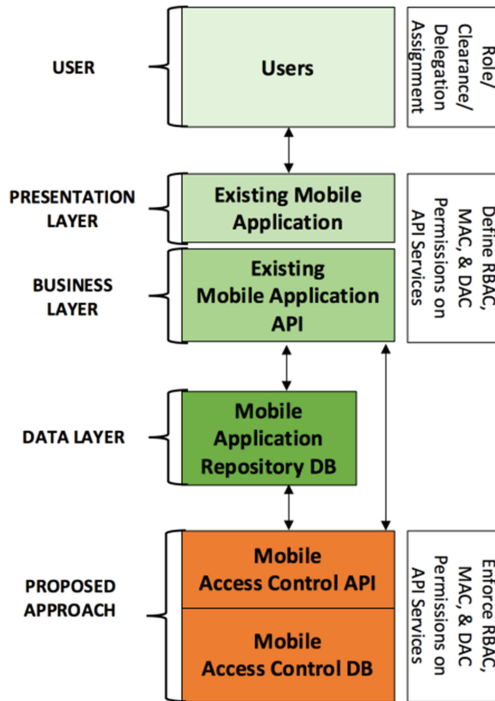


Fig. 2. Intercepting API calls approach architecture.

To illustrate the third group, Fig. 3 details the modifications of the original API services that are needed for interception. Specifically, for a mobile app, there is a set of original mobile app API services, as shown in the left side of Fig. 3. To maintain the

functionality of the mobile app and provide an ability to continue to invoke services by name, the original mobile app API services are renamed (as shown on the right side of Fig. 3) in order to reuse the original name of the original service for the new intercepting API services so that the original services from the mobile app remain unchanged (would now be occurring against the intercepting services). For each original mobile app API service, we define a corresponding intercepting API service, as shown in the bottom part of Fig. 3, that is able to: perform RBAC, MAC, and DAC security checks for the user/role/clearance combination; call the corresponding mobile app API service (if it is allowed); and then return either filtered data (retrievals) or success/failure (inserts, updates, or deletes) status.

The mobile app is still able to invoke the same original API services by name and signature, which are now the intercepting API services (with the same signature) that are able to step in and interrupt the process when they are called/invoked. As a result, the intercepting API services act as a wrapper that adds a security layer to the original API services. The dashed arrows in Fig. 3 indicate that the process of renaming the original API services as well as the process of creating the intercepting file needs to be done only once. Therefore, the developer only needs to create these files once and after that security administrators can manage the RBAC, MAC, and DAC policies without modifying the server-side portion of the mobile app through the means of a separate user interface. The solid arrow indicates the way that the API behaves when a user makes a request through the mobile app; first, the request is intercepted in order to be evaluated with the pertinent access control policies and then, depending on the result, we either proceed to execute the request or send an error message to the user who sent the request.

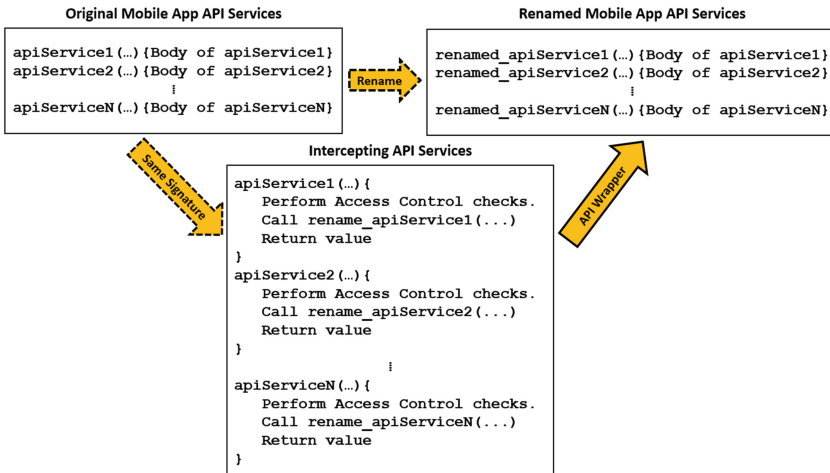


Fig. 3. Conceptual API process [16].

4 A Model for Services-Based RBAC, MAC, and DAC

This section presents a model for services-based RBAC, MAC, and DAC, by discussing the way that the API of a mobile app is viewed from a security perspective in order to control who can invoke which service(s) of an API at which times, and the way that each service is viewed from a security standpoint. The model presented upgrades traditional object-based permission of RBAC, MAC, and DAC to be service based. In support of this process, we categorize the services on an API in different ways. The idea behind the Intercepting API Calls approach is to secure highly-sensitive information that is present in mobile applications and is accessible via an API. To support this focus, we assume that data transactions between a mobile app and a server are performed via an API. Through this mobile app API, we seek to provide a means for a user playing a role, and possibly contain a clearance, to be constrained to deliver/store data when utilizing the mobile app via the interception of the invocations of API services. The Intercepting API Calls approach makes use of the ability to define permissions on the services of the API in three different ways. First, the API can be partitioned into Secure/Unsecure services (Defn. 4) where the Secure services can be assigned on a role-by-role basis (user's role in Defn. 1), thereby supporting RBAC. Not all of the API services need to be in the secure category; for example, API services to load drop downs, display web content, etc., may not need to be secure. The Secure API services are the ones that lead to data that is stored/edited/displayed that must be controlled by role. Second, the API can be partitioned into Labeled/Unlabeled services (Defn. 5) where each Labeled service has a classification and Labeled services can be assigned based on a user's clearance (Defns. 2v2 and 3), thereby supporting MAC. Similar to secure services, Labeled services are a subset of the API that require control from a security perspective and can be assigned to classifications. As mentioned in the RBAC perspective for the secure category, not all of the API services need to be in the labeled category. The Labeled API services are the ones that lead to data that is stored/edited/displayed that must be controlled by classifications. Unlabeled services need not be assigned and are available to any user. Third, if an API is partitioned by using either RBAC or MAC, an original user (Defn. 6) or a delegated user with pass-on delegation authority (Defns. 7 and 8) can delegate a full (Defns. 9 and 11) or a partial (Defns. 10 and 12) set of their services to a delegated user.

Defn. 1: A role r is defined as a two-pair $r = \langle r_{ID}, r_{Name} \rangle$ with unique identifier r_{ID} and name r_{Name} .

Example: The CT² app has several roles, one of which would be for a parent: $r = \langle r_{ID4}, Parent \rangle$.

Defn. 2: A user u is defined as a tuple $\langle u_{ID}, u_{Name}, u_{CLR} \rangle$, with unique u_{ID} identifier, name u_{Name} and optional clearance $u_{CLR} \in \{TS, S, C, U\}$ that signifies that a user is limited to information (UI) in the GUI by MAC and services (API) that satisfy the established MAC properties (e.g., simple integrity, simple security, liberal *, strict *, etc.).

Example: The CT² app has a user with top secret clearance $\langle u_{ID1}, Karen, TS \rangle$.

Defn. 3: A user u that has a clearance u_{CLR} (Defn. 2) assigned has also a read property and a write property assigned to control access to a service α as follows:

Read Properties:

- Simple Security (SS-r): User u has read access on service α iff $u_{CLR} \geq \alpha_{CLS}$.
- Strict * (Read) (S*-r): User u has read access on service α iff $u_{CLR} = \alpha_{CLS}$.

Write Properties:

- Simple Integrity (SI-w): User u has write access on service α iff $u_{CLR} \geq \alpha_{CLS}$.
- Strict * (Write) (S*-w): User u has write access on service α iff $u_{CLR} = \alpha_{CLS}$.
- Liberal * (L*-w): User u has write access on service α iff $u_{CLR} \leq \alpha_{CLS}$.

Given Defn. 3, we revise Defn. 2 as below:

Defn. 2 v2: A user u is defined as a tuple $\langle u_{ID}, u_{Name}, u_{CLR}, u_{MACRD}, u_{MACWR} \rangle$, where $u_{MACRD} \in \{SS, S^*\}$ and $u_{MACWR} \in \{SI, S^*, L^*\}$.

Updated Example: The CT² app has four users each with different clearances and read/write properties: $U_{CT}^2 = \{u_1 = \langle u_{ID1}, Karen, TS, SS-r, L^*-w \rangle, u_2 = \langle u_{ID2}, Carmen, C, S^*-r, S^*-w \rangle, u_3 = \langle u_{ID3}, Joe, C, SS-r, S^*-w \rangle, u_4 = \langle u_{ID4}, Peter, S, SS-r, S^*-w \rangle\}$, where: nurse Karen can read down and write up and has the most privileges, parent Carmen is limited to one level secret; and, coach Joe and AT Peter can both read down and write equal.

Defn. 4: The API β_{MA} of a mobile application MA can be partitioned into two disjoint sets Secure API σ_β and Unsecure API μ_β in regards to the services that are to be assigned by role:

- Secure API $\sigma_\beta \subseteq \beta_{MA}$ are the services of MA that need to be controlled.
- Unsecure API $\mu_\beta \subseteq \beta_{MA}$ are the services of MA that do not need to be controlled where $\beta_{MA} = \sigma_\beta \cup \mu_\beta$ and $\sigma_\beta \cap \mu_\beta = \emptyset$ (e.g., $\mu_\beta = \beta_{MA} - \sigma_\beta$).

Example: The following services are utilized in the API that provides and stores data for the CT² mobile application:

- GET/states – Gets the list of states available
- POST/concussions/followup/add/{concussionEventId} – Inserts follow up data of a student into the database

The first service stated above does not need to be secured since all of the users of the mobile application can view the list of states (this is not

Table 1. Secure/Unsecure services of CT².

Secure/Unsecure	Service name
Secure	GET/user/:userId
Secure	GET/userAccounts/account/:userId
Secure	GET/useraccounts/:username/:password
Secure	GET/userRoleSchool/:userid
Secure	POST/userAccounts/add
Secure	GET/students/school/:schoolId
Secure	GET/student/:studentId
Secure	GET/students/:firstName/:lastName
Secure	GET/student/guardians/:studentId
Secure	POST/students/add
Secure	POST/students/update/:studentId
Secure	POST/students/guardian/add
Secure	POST/students/guardian/update/:guardianId
Secure	GET/concussion/:concussionId
Secure	GET/concussion/followups/:concussionId
Secure	GET/concussion/followup/symptoms/:recordId
Secure	GET/concussions/school/:schoolId
Secure	GET/concussions/student/:studentId
Secure	GET/concussions/user/:userId
Secure	GET/concussions/status/:incidentId/:status
Secure	POST/concussions/add
Secure	POST/concussions/update/:incidentId
Secure	POST/concussions/followup/add/ :concussionEventId
Secure	POST/concussions/followup/update/ :followUpId/:referenceId+
Secure	GET/concussion/symptoms/:referenceId+
Secure/Unsecure	Service name
Unsecure	GET/states
Unsecure	GET/regions/:stateId
Unsecure	GET/districts/:regionId
Unsecure	GET/schools/all
Unsecure	GET/schools/:districtId
Unsecure	GET/schools/:schoolId
Unsecure	GET/menu/assessmentTools
Unsecure	GET/menu/eventLocations
Unsecure	GET/menu/contactMechanisms
Unsecure	GET/menu/medicalimaging
Unsecure	GET/menu/diagnosingroles
Unsecure	GET/menu/headLocation
Unsecure	GET/menu/sports
Unsecure	GET/menu/symptoms
Unsecure	GET/menu/symptoms/within
Unsecure	GET/menu/symptoms/lingering
Unsecure	GET/menu/roles

confidential data), nonetheless, the second service needs to be secured since only a subset of the roles available are allowed to add students' follow up data. Table 1 summarizes the secure/unsecure services that are partitioned from the REST API of CT².

Defn. 5: The API β_{MA} of a mobile application MA can be partitioned into two disjoint sets Labeled API δ_β and Unlabeled API θ_β in regards to the services that are to be controlled by classifications:

- Labeled API $\delta_\beta \subseteq \beta_{MA}$ are the services of MA that need to be controlled.
- Unlabeled API $\theta_\beta \subseteq \beta_{MA}$ are the services of MA that do not need to be controlled where $\beta_{MA} = \delta_\beta \cup \theta_\beta$ and $\delta_\beta \cap \theta_\beta = \emptyset$ (e.g., $\theta_\beta = \beta_{MA} - \delta_\beta$).

Example: The service $GET/concussion/followups/:concussionId$ and the service $POST/concussions/followup/add/:concussionEventId$ can be placed in both the secure API set and the labeled API set since these services retrieve/add highly-sensitive data. Table 2 summarizes the labeled/unlabeled services that are partitioned from the REST API of CT².

Note that a labeled service can have a sensitivity level of unclassified. In MAC, data often moves from level to level, so what is unclassified today, could be confidential or secret at a later point in time; this could be true of services. Only the labeled services presented in Table 2 have classifications as shown in Table 3. The unlabeled services in Table 2 are all related to the display of menu drop down values, selection values, etc.

Defn. 6: An *original user*, ou , is a user that owns a given role.

Defn. 7: A *delegated user*, du , is a user to whom a role will be delegated.

Defn. 8: *Pass On Delegation Authority (PODA)* is a Boolean value assigned to a user which determines if he/she can delegate his/her permissions to another user ($poda = true$) or not ($poda = false$).

Example: In the CT² app, Nurse Karen can be an original user, ou , with the original role, or , nurse. If a school needs substitute nurse Lois to cover for Karen, Karen could delegate her original role or to Lois as the delegated user du and can also authorize Lois to delegate the delegated role further ($poda = true$).

Defn. 9: A *Full RBAC Services (FRS) Delegation* $d_{FRS} = \langle ou, or, du, dr, \varphi, poda, timePeriod \rangle$ delegates all of the assigned secure service permissions $\varphi \in r_p$ from an original user, ou , with an original role, or , to a delegated user, du , with a delegated role $dr = or$ with the potential to pass on ($poda$ is true or false), and $timePeriod = \{startTime, endTime\}$ which represents the period of time in which the du has access to the delegated permissions.

Example: The original user ou Karen $\langle u_{ID1}, Karen, TS, SS-r, L^*-w \rangle$ seeks to delegate original role or nurse and all of the secure assigned

Table 2. Labeled/Unlabeled services of CT².

Labeled/Unlabeled	Service name
Labeled	GET/user/:userId
Labeled	GET/userAccounts/account/:userId
Labeled	GET/useraccounts/:username/:password
Labeled	GET/userRoleSchool/:userid
Labeled	POST/userAccounts/add
Labeled	GET/students/school/:schoolId
Labeled	GET/student/:studentId
Labeled	GET/students/:firstName/:lastName
Labeled	GET/student/guardians/:studentId
Labeled	POST/students/add
Labeled	POST/students/update/:studentId
Labeled	POST/students/guardian/add
Labeled	POST/students/guardian/update/:guardianId
Labeled	GET/concussion/:concussionId
Labeled	GET/concussion/followups/:concussionId
Labeled	GET/concussion/followup/symptoms/:recordId
Labeled	GET/concussions/school/:schoolId
Labeled	GET/concussions/student/:studentId
Labeled	GET/concussions/user/:userId
Labeled	GET/concussions/status/:incidentId/:status
Labeled	POST/concussions/add
Labeled	POST/concussions/update/:incidentId
Labeled	POST/concussions/followup/add/ :concussionEventId
Labeled	POST/concussions/followup/update/ :followUpId/:referenceId+
Labeled	GET/concussion/symptoms/:referenceId+
Labeled/Unlabeled	Service name
Unlabeled	GET/states
Unlabeled	GET/regions/:stateId
Unlabeled	GET/districts/:regionId
Unlabeled	GET/schools/all
Unlabeled	GET/schools/:districtId
Unlabeled	GET/schools/:schoolId
Unlabeled	GET/menu/assessmentTools
Unlabeled	GET/menu/eventLocations
Unlabeled	GET/menu/contactMechanisms
Unlabeled	GET/menu/medicalimaging
Unlabeled	GET/menu/diagnosingroles
Unlabeled	GET/menu/headLocation
Unlabeled	GET/menu/sports
Unlabeled	GET/menu/symptoms
Unlabeled	GET/menu/symptoms/within
Unlabeled	GET/menu/symptoms/lingering
Unlabeled	GET/menu/roles

services of her Nurse role the delegated user *du* Lois, a substitute school nurse for one day: $del = \langle u_{ID1}, r_{ID3}, u_{ID5}, r_{ID3}, \varphi, false, \{2017-07-31T09:00:00+00:00, 2017-12-15T07:00:00+00:00\} \rangle$.

Defn. 10: A *Partial RBAC Services (PRS) Delegation* $d_{PRS} = \langle ou, or, du, dr, \varphi', poda, timePeriod \rangle$ delegates a subset of the assigned secure service permissions $\varphi' \in r_p$ and $\varphi' \subseteq \varphi$ from an original user, *ou*, with an original role, *or*, to a delegated user, *du*, with a $dr = or$ with the potential to pass on (*poda* is true or false) and $timePeriod = \{startTime, endTime\}$ which represents the period of time in which the *du* has access to the delegated permissions.

Example: The original user *ou* Karen $\langle u_{ID1}, Karen, TS, SS-r, L^*-w \rangle$ seeks to delegate original role *or* nurse and only a subset of the secure services assigned to her Nurse role the delegated user *du* Lois, a substitute school nurse for one day to only log on and be able to read (GET services) information on students: $del = \langle u_{ID1}, r_{ID3}, u_{ID5}, r_{ID5}, \varphi', false, \{2017-07-31T09:00:00+00:00, 2017-12-15T07:00:00+00:00\} \rangle$, where $\varphi' = \{GET/user/:userid, \dots, GET/userRoleSchool/:userid, POST/userAccounts/:add, \dots, GET/student/guardians/:studentId\}$.

Table 3. Classifications for labeled services of CT².

Classification	Service name
Confidential	GET/user/:userId
Confidential	GET/userAccounts/account/:userId
Confidential	GET/useraccounts/:username/:password
Confidential	GET/userRoleSchool/:userid
Top Secret	POST/userAccounts/add
Confidential	GET/students/school/:schoolId
Confidential	GET/student/:studentId
Confidential	GET/students/:firstName/:lastName
Confidential	GET/student/guardians/:studentId
Confidential	POST/students/add
Top Secret	POST/students/update/:studentId
Confidential	POST/students/guardian/add
Confidential	POST/students/guardian/update/:guardianId
Confidential	GET/concussion/:concussionId
Secret	GET/concussion/followups/:concussionId
Secret	GET/concussion/lookup/symptoms/:recordId
Confidential	GET/concussions/school/:schoolId
Confidential	GET/concussions/student/:studentId
Confidential	GET/concussions/user/:userId
Confidential	GET/concussions/status/:incidentId/:status
Confidential	POST/concussions/add
Top Secret	POST/concussions/update/:incidentId
Secret	POST/concussions/lookup/add/:concussionEventId
Top Secret	POST/concussions/lookup/update/:followUpId/:referenceId+
Secret	GET/concussion/symptoms/:referenceId+

For MAC, if we choose to delegate, then we are delegating a combination of the clearance of the user and the read and write properties for the user; this allows the delegated user to access the appropriate labeled services by classification. In *Full MAC Services Delegation*, a user delegates all of his/her labeled services permissions (user/CLR/read-write properties) to a delegated user. In *Partial MAC Services Delegation*, a user delegates his/her read-write properties and a CLR that is less secure than his current level, thereby automatically resulting in a subset of methods that is at most the same of the original clearance level but is more often less.

Defn. 11: A *Full MAC Services (FMS) Delegation* $d_{FMS} = \langle ou, oclr, oprops, du, dclr, dprops, \Omega, poda, timePeriod \rangle$ delegates all of the assigned labeled service permissions $\Omega \in u_p$ from an original user, ou , with an original clearance, $oclr$, and original read/write properties, $oprops$, to a delegated user du with a delegated clearance $dclr = oclr$ and delegated read/write properties, $dprops = oprops$ with the potential to pass on ($poda$ is true or false) and $timePeriod = \{startTime, endTime\}$ represents the period of time in which the du has access to the delegated permissions.

Example: The original user ou Karen $\langle u_{ID1}, Karen, TS, SS-r, L^*-w \rangle$ seeks to delegate her MAC privileges (CLR and read/write properties) and all of the labeled services she has permission to access to the delegated user du Lois, a substitute school nurse for one day: $del = \langle u_{ID1}, TS, SS-r, L^*-w, u_{ID5}, TS, SS-r, L^*-w, \Omega, false, \{2017-07-31T09:00:00+00:00, 2017-12-15T07:00:00+00:00\} \rangle$.

Defn. 12: A *Partial MAC Services (PMS) Delegation* $d_{PMS} = \langle ou, oclr, oprops, du, dclr, dprops, \Omega', poda, timePeriod \rangle$ delegates a subset of the assigned labeled service permissions $\Omega' u_p \subseteq \Omega$ and $\Omega' \in u_p$ from an original user, ou , with an original clearance, $oclr$, and original read/write properties, $oprops$, to a delegated user du with a delegated clearance $dclr = oclr$ and delegated read/write properties, $dprops = oprops$ with the potential to pass on ($poda$ is true or false) and $timePeriod = \{startTime, endTime\}$ represents the period of time in which the du has access to the delegated permissions.

Example: The original user ou Karen $\langle u_{ID1}, Karen, TS, SS-r, L^*-w \rangle$ seeks to delegate her MAC privileges (CLR and read/write properties) and only GET services she has permission to the delegated user du Lois, a substitute school nurse for one day: $del = \langle u_{ID1}, TS, SS-r, L^*-w, u_{ID5}, C, SS-r, L^*-w, \Omega', false, \{2017-07-31T09:00:00+00:00, 2017-12-15T07:00:00+00:00\} \rangle$, where $\Omega' = \{GET/user/:userid, GET/userAccounts/account/:userid, \dots, GET/concussion/symptoms/:reference\ ID\}$.

Intercepting calls for unsecure and unlabeled services are automatically passed through since there are no required security checks. A given mobile application can have a partitioning of the API into: Secure/Unsecure in support of RBAC, Labeled/Unlabeled in support of MAC, or both. In addition, the Intercepting API Calls approach requires minimal or no changes to the UI of the mobile application other than for the need to identify a given role/clearance for a session being initiated by a user.

In this case, we incorporate the functionality of original API services into REST or API services that are utilized to intercept the original API calls to disable the delivery of content to the user.

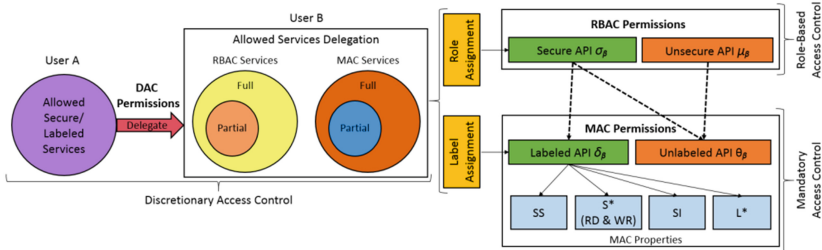


Fig. 4. RBAC, MAC, and DAC permissions for API services.

Figure 4 conceptualizes the RBAC, MAC, and DAC permissions associated with services. The secure/unsecure services from Defn. 4 and Table 1 are assigned by role with the user acquiring these services when they choose a role for a given session (a user may have multiple roles but is limited to one role per mobile application session). This allows RBAC to be used to control services independent of MAC. The labeled/unlabeled services from Defn. 5 and Table 2 are accessible based on a user’s clearance that dominates the classification of the services under the properties (simple security, liberal-*, etc.). This allows MAC to control services independent of RBAC. In addition, we can control services for security policies of users that have both RBAC and MAC. This can be achieved by classifying the services of an API as secure/unsecure in support of RBAC and then extending security by classifying secure services even further as labeled/unlabeled services in support of MAC. Note that secure services can be classified as either labeled/unlabeled but unsecured services can only be classified as unlabeled services (dashed arrows shown in Fig. 4). The left side of Fig. 4 depicts the way that API services can be delegated. The services can be delegated by an original user (Defn. 6) or a delegated user (Defn. 7) with pass on delegation authority (represented as User A in Fig. 4) in four different ways: *Full RBAC Services (FRS) Delegation* in Defn. 9 to delegate all of his/her allowed Secure services to a delegated user (biggest circle on the left within User B box in Fig. 4); *Partial RBAC Services (PRS) Delegation* in Defn. 10 to delegate a portion of his/her allowed Secure services to a delegated user (smallest circle on the left within User B box in Fig. 4); *Full MAC Services (FMS) Delegation* in Defn. 11 to delegate all of his/her allowed Labeled services to a delegated user (biggest circle on the right within User B box in Fig. 4); and, *Partial MAC Services (PMS) Delegation* to delegate a portion of his/her allowed Labeled services to a delegated user (smallest circle on the right within User B box in Fig. 4).

5 An RBAC/MAC/ DAC Approach for Intercepting API Calls

This section discusses the interactions and infrastructure of the Intercepting API Calls approach for RBAC, MAC, and DAC. To begin, Fig. 5 depicts the detailed interactions of the approach. The steps from the user's perspective from left to right are: login to his/her mobile app account; for successful login, extract the user's role/clearance that is part of the login credentials; store the extracted user role/clearance in a secure access token in order to use it in future API calls; utilize the mobile app which results in multiple mobile app API calls and are intercepted (data processing in top of Fig. 5); and, the intercepted API call interacts with the access control permissions and policies to enforce the defined security before invoking the original mobile app API call. There are two possible requests that can occur as an end result of the interactions: *insert/update/delete requests* where the data that the user is trying to insert/update/delete is not allowed if the user/role/clearance combination does not have permission to do so; and, *retrieve requests* where the data that the user/role/clearance combination is trying to retrieve is filtered according to his/her role/clearance.

In the insert/update/delete request (via an intercepting mobile app API call in the upper portion of the Access Control API oval in Fig. 5), the request is intercepted to perform the access control checks, and depending on the response, the action is either done (the original mobile app API call is allowed) or not. In the retrieve request the user is trying to retrieve data (via an intercepting mobile app API call in the lower portion of the Access Control API oval in Fig. 5), the data source performs this action but the mobile app API is intercepted to allow access control checks to be performed. This allows the intercepted API call to determine if the user has access to all/some/none parts of the data with the resulting original API call returning data (all/some case) or null/error message (none case).

To manage which resources a specific role and/or clearance can access, we store the access control policy in a database, represented in Fig. 6 as an entity-relationship diagram to support our service-based RBAC, MAC, and DAC approach. Once the user's role and/or clearance has been verified, we can access the specific permission we want to evaluate through the means of an API service as stated in the previous paragraph. The database would hold the roles and/or clearances for each user of each mobile app along with the permissions for each role and/or clearance to each HIT system supported with the HAPI FHIR server. Specifically, to track which services of which FHIR RESTful APIs for each HIT are authorized by role and/or clearance to a user of a particular mHealth app. Moreover, the *secure_unsecure_services* and the *labeled_unlabeled_services* entities provides details of whether a user has access to the resource he/she requested or not by his/her role or clearance, respectively. In addition, the security policy tables store information about the available CRUD services, resources, roles (RBAC), clearances (MAC), classifications (MAC), read and write constraints (MAC), and delegations (DAC).

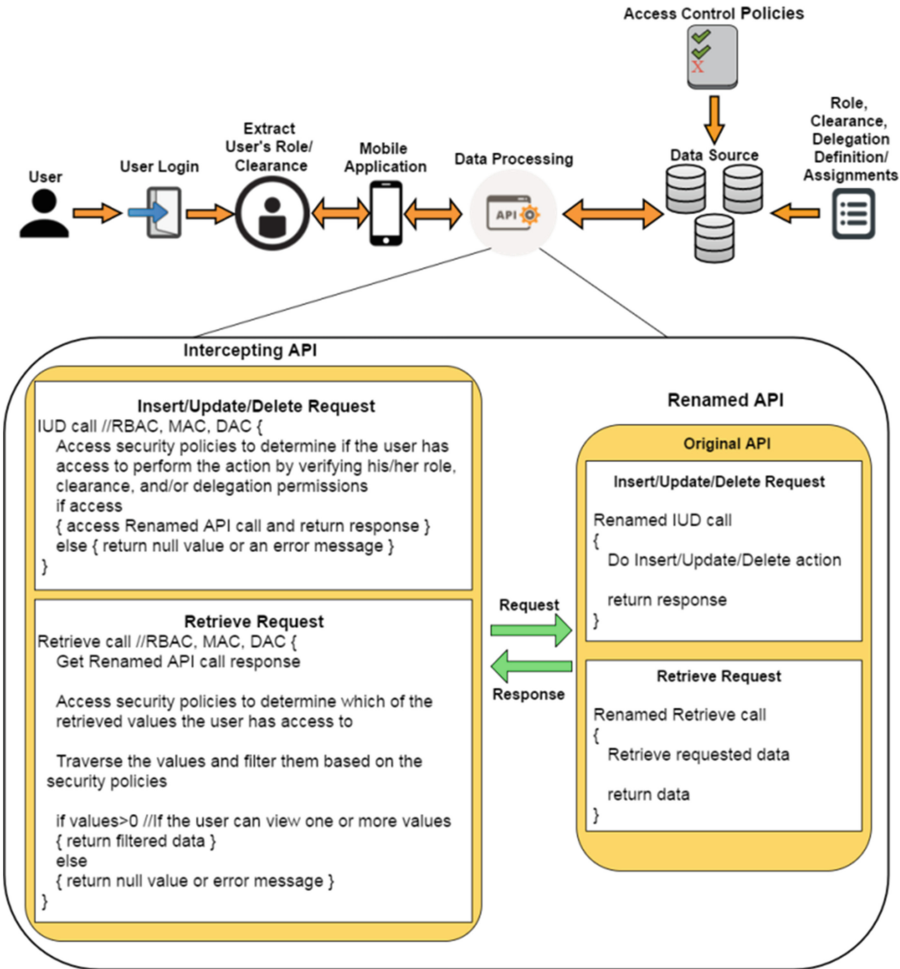


Fig. 5. Interactions for the intercepting API calls approach.

6 RBAC/MAC/DAC Intercepting API Calls Enforcement Code

This section reviews the algorithm that is able to automatically generate the Intercepting API Calls enforcement code for RBAC, MAC, and DAC permission checks that can occur dynamically and in real time. The primary changes to support the Intercepting API Calls approach are made in the backend of the mobile app (server-side – bottom portion of Fig. 2) and include the addition of RBAC, MAC, and DAC security policies in a permission database to create the mapping from the original mobile app.

API services to the corresponding new intercepting API services as shown in Fig. 6. Each new intercepting API service has the same signature (same address and parameter) as its original counterpart, so that the intercepting API call can substitute for the original API call of the mobile app to allow the aforementioned security checks for retrieve and insert/update/delete requests. As a result, the Intercepting API Calls approach effectively wraps the original mobile app calls. The mobile app now seamlessly invokes the intercepting API calls. These Intercepting API Calls enforcement code contains the appropriate RBAC, MAC, and DAC security checks, adding a layer of security to enforce the policies. The renamed original mobile app API services are invoked based on the outcomes of the security checks. The end result is that the mobile app appears differently based on the user/role combination, to limit information that is delivered (retrieve request) or that impacts the data that is stored (insert/update/delete requests).

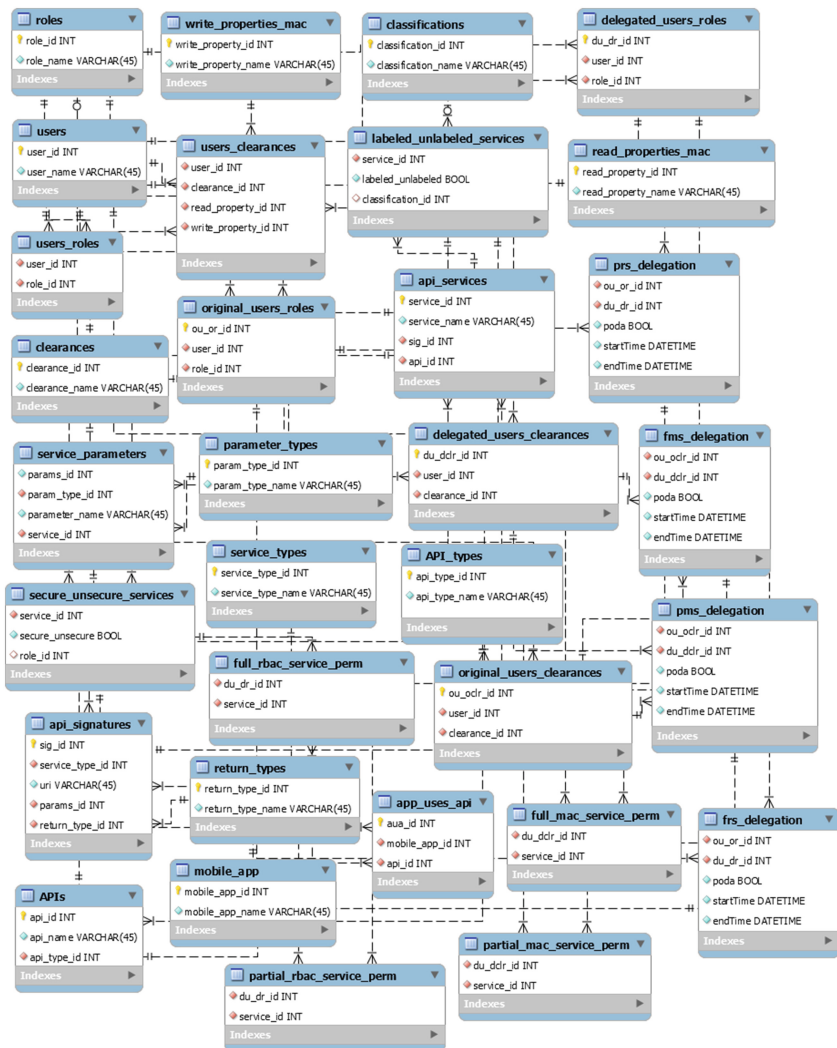


Fig. 6. ER diagram for supporting the intercepting API calls approach.

The Intercepting API Calls approach utilizes an algorithm to automatically generate the intercepting code. Pseudo-code for the algorithm is shown in Fig. 7. In order to automatically generate the enforcement code for the approach, we need to create a file that contains the same API services as the original mobile app API via the generate function `Access_Control_API_Generator` which has a parameter that contains an array of all the API services available in the mobile application (line 1 of Fig. 7). For each of the API services in the array, we obtain the parameters (if any), which are stored in a database and store these (line 3 of Fig. 7) in a variable. Once we obtain the parameters of the API call that is being evaluated, we can generate the heading of the intercepting API call function by using the current API call as well as its parameters (if any) (line 4 of Fig. 7). After generating the heading for the intercepting API call function, we then generate the body of the API call, which contains the security policies for that specific call and invokes the original mobile app API call if the user has access to it (line 6 of Fig. 7). The resulting heading and body of the current API call is stored in an array (line 7 of Fig. 7). Once all of the intercepting calls have been created, we traverse the array in which they are stored in order to generate the intercepting file (line 9 of Fig. 7).

```

1 Access_Control_API_Generator(API_Calls){
2   foreach(API_Calls as currentAPICall){
3     params = getParams(currentAPICall);
4     API_Call_Heading = generateHeading(currentAPICall, params);
5     /*API_Call_Body - Contains security policies and a call to the
        original API service.*/
6     API_Call_Body = generateBody(currentAPICall);
7     API_Calls_Array = insert(generateAPICall(API_Call_Heading,
                                                API_Call_Body));
8   }
9   GenerateFile(API_Calls_Array);
10}

```

Fig. 7. Pseudo code algorithm to generate enforcement code of intercepting API calls approach.

To demonstrate the algorithm in Fig. 7, Fig. 8 contains the actual PHP enforcement code that we implemented in order to generate our approach to the API of the CT² mobile app. The function presented in Fig. 8 is utilized to generate the services in the intercepting API. In order to create a renamed original API service for each of the original mobile app API services, we need the name of the service we are going to generate, if the service needs to be secured by adding permissions and, the name of the file in which we add the generated service (line 1 of Fig. 8). Note that the permissions we add in each of the intercepting services (if needed) are a layer of security that is not part of the original API services (lines 3–13 of Fig. 8). Basically, there are three different types of security permissions we can enforce: permissions based on a user's role, permissions based on a user's clearance and MAC properties and, permissions based on delegations. To verify if the user has access to the requested API service, we access the security policy stored in the database which contains the entities `secure_unsecure_services`, `frs_delegation`, `prs_delegation`, `labeled_unlabeled_services`, `fms_delegation`, `pms_delegation` entities shown in Fig. 6 of Sect. 6 that specify the

requested API service's role/classification (lines 3–7 and lines 11–13 of Fig. 8). If the role/clearance that is been verified does have permission to perform the requested action, then the service proceeds to access the service in the renamed API file (lines 8–10 of Fig. 8); otherwise, the intercepting API service returns a null value (line 12 of Fig. 8). Nonetheless, if the renamed API service does not need to verify a user's role in order to be executed then the intercepting call invokes it directly, in other words, the intercepting API service does not add security permissions in this case (lines 8–10 of Fig. 8). Finally, the generated API service gets written in the file that serves as the intercepting API (line 15 of Fig. 8).

The code given in Fig. 8 generates, for each original services of the CT² API, a REST API for generating an intercepting API file in support of the Intercepting API Calls approach. This is shown in Fig. 9 for the original CT² API service *updateStudent* while Fig. 10 shows the renamed CT² API of the aforementioned service.

```

1 function echoInterceptBody($serviceName, $need_permission,
  $write_file){
2   $wrapper_string = "public function ".$serviceName."{";
3   if($need_permission){
4     $wrapper_string = $wrapper_string."
5       \ $permission = \ $this->verifyAPIPermissions(__FUNCTION__);
6     if(\ $permission == 1){";
7   }
8   $wrapper_string = $wrapper_string."
9     \ $renamedConcussionUConn = new renamedConcussionUConn();
10    return \ $renamedConcussionUConn->RENAMED".$serviceName."";";
11 if($need_permission){
12   $wrapper_string = $wrapper_string.") else{return NULL;};";
13 }
14 $wrapper_string = $wrapper_string."}";
15 fwrite($write_file, $wrapper_string);
16 }

```

Fig. 8. Code for generating the body of the API services in the CT² API.

```

1 public function updateStudent($studentObject, $studentId){
2   $permission = $this->verifyAPIPermissions(__FUNCTION__);
3   if($permission == 1){
4     $renamedConcussionUConn = new renamedConcussionUConn();
5     return $renamedConcussionUConn->
6       RENAMEDupdateStudent($studentObject, $studentId);
7   } else{return NULL;}}

```

Fig. 9. Portion of generated code for the intercepting API.

```

Public function RENAMEUpdateStudent($studentObject,$studentId){
    $sqlGeneralStudent = "UPDATE students SET first_name =
        \' . $studentObject->firstName . \',
        middle_name = \' . $studentObject->middleName . \',
        last_name = \' . $studentObject->lastName . \',
        suffix = \' . $studentObject->suffix . \',
        email = \' . $studentObject->email . \',
        student_number = \' . $studentObject->studentNumber . \',
        school_id = \' . $studentObject->schoolId . \'
    WHERE student_id = \' . $studentId;
    $recordId = $this->updateRecord($sqlGeneralStudent);
    if($recordId){
        $sqlStudentDemo="UPDATE student_demographics date_of_birth = \' .
            $studentObject->dateOfBirth . \',
            gender = \' . $studentObject->gender . \'
        WHERE student_id = \' . $studentId;
        if($this->updateRecord($sqlStudentDemo)) return 1;
        else return 2;
    }
    else{return 0;}
}

```

Fig. 10. Portion of generated code for the renamed API.

7 Implementation of RBAC/MAC/DAC Intercepting API Calls

To evaluate the Intercepting API Calls approach, the Connecticut Concussion Tracker (CT²) mobile application, database, and its server are utilized as an example. As currently designed, the CT² app supports RBAC, MAC, and DAC that allows for the different screens and the content of different screens to be available by role, clearance, and delegations. There are four roles: the *Nurse* role, which has access to all tabs for a school nurse to manage a student's concussion incident from its occurrence to its resolution; the *Athletic Trainer (AT)* role which has access to home, list, student, cause, and symptoms tabs to do a limited preliminary assessment if a concussion incident occurs at the event; the *Coach* role, which has access to home, list, student and cause tabs to report a concussion incident at an athletic event with very limited information on the student; and, the *Parent* role, which has access to home, list, student, cause, and symptoms tabs to both report a concussion incident on his/her child while attending the athletic event or to track the current status of his/her children that have ongoing concussions. In addition, each of the users of the mobile app have a clearance assigned and delegation permissions.

Programmatically, we have source code for the Android version of the CT² app and a REST API that accesses the concussion MySQL database. The source code of the mobile app is organized by tabs that are loaded for a given user/role combination, and each tab is augmented with if/else conditions that either display the data on a tab if it was available in the database or display an error message stating that the contents couldn't be retrieved. The realization of the Intercepting API Calls approach is achieved without any modification to the mobile app UI and is intended to allow fine-grained access control on the information that is displayable and/or storable of the authorized tabs for each user/role/clearance combination. There is a very clear mapping

from the process described in this section and the accompanying figures to its realization in CT². The database is augmented with a table that contains a list of all the API services available along with a service_id, and tables that contain the security policies that determine which invocations the available roles/clearances have access to (secure_unsecure_services and labeled_unlabeled_services entities in Fig. 6 of Sect. 5). Given these database changes, we then take the original CT² REST API services and rename as shown in Fig. 3. Afterwards, a set of new CT² intercepting REST API services are defined that perform a series of RBAC, MAC, and DAC checks and if successful, invoke the corresponding renamed original CT² REST API services.

From a process perspective, the steps follow the top portion of Fig. 5. The user logs on to the CT² mobile app and his/her role/clearance is stored in a global variable in order to support the class that manages the API services. Figure 11 illustrates the impact of the Intercepting API Calls and associated process for a user with the role of *Coach* and with a clearance of Confidential (C) which has access to only the home, list, student, and cause tabs. This role-clearance combination can add basic information on the ‘Student’ tab and can add information in the ‘Cause’ tab and, after adding the information, can view but not edit. The original mobile app CT² API services support the insert of information in the database and the intercepting API call in this case allows that first save to occur. At a later point in time, if the user attempts to edit and perform another save, the intercepting API call in this case, performs the access control check that does not allow the edit. As a result, when a user with the role of Coach and with a clearance of C that is using the ‘Cause’ tab attempts to save, the intercepting API call alerts that he/she does not have permission to perform that action. The other tabs of

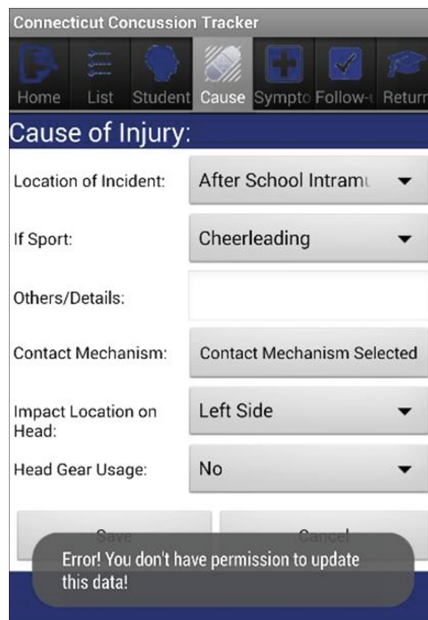


Fig. 11. ‘Cause’ screen for the role of *Coach* in CT² [16].

CT², ‘Symptoms’, ‘Follow-Up’ and ‘Return’, are still visible within the app. However, when a user with the Coach role and with a clearance of C attempts to access one of these tabs, the application tries to obtain the pertinent data via the former original CT² API call that has been replaced by a new CT² intercepting API call that checks for permissions and returns that the specified role-clearance combination does not have permission to retrieve the data for those screens.

8 Related Work

There are many efforts that propose access control mechanisms to secure mobile applications by limiting the permissions and resources a mobile app can access in different areas of the mobile device/app. In this section, we discuss several existing approaches that attempt to apply access control mechanisms on different locations on a mobile device and, we explain the way our approach compares and contrasts. The first area of related work involves sensor management on smartphones that is commonly addressed by applying access control mechanisms to the sensors of a mobile device so that mobile apps obtain fine-grained permissions. This facilitates the managing of sensor data in mobile apps (e.g., user’s location, use of Bluetooth) [32, 33]. BlurSense [32] and SemaDroid [33] allow users to define and add privacy filters to sensor data, through the means of a user interface, that is being used on their mobile applications. In contrast to these efforts, our work presented in this paper focuses on API access control management for the API services that are utilized within a mobile app to populate data in the app and to add/edit data and store it in a data source. In other words, instead of focusing on modifying the operating system to filter sensor data we modify the backend of a specific mobile app and filter the data that a user can have access to according to his/her role/clearance/delegation permissions, which can include sensor data as well if there was an API service included in the intercepted API that managed this.

The second area of related work involves permission control in Android in which access control can be applied on the mobile device itself. There are many existing approaches [34–39] that focus on applying fine-grained access control policies to mobile devices that contain Android as their operating system. This is due to the fact that Android contains a coarse-grained access control mechanism when it comes to allowing permissions in mobile applications. In other words, in order for a user to install a mobile app he/she needs to accept all of the permissions that the app requires. This may disregard the fact that some permissions may not be necessary for the app to function and that some of the permissions may not make sense for the app that is being downloaded and could result in using the allowed component for malicious purposes (e.g., a flashlight app tells user it needs permission to get the user’s location). Adding fine-grained access control to the APIs that Android uses for the device and apps to function properly has been addressed by: mocking the values that an app receives in order to function [34] (e.g., mocking latitude and longitude coordinates); extending the security policies of the mobile device [35–37]; by rewriting the bytecode of the mobile device [38]; and by adding security modules to the mobile device [39]. Moreover, another effort by [40] utilizes MAC to provide security in mobile computing by proposing and implementing *FlaskDroid*, a security architecture that provides

mandatory access control in both middleware and kernel layers of Android OS. The purpose of this work is to apply fine-grained MAC security policies to Android OS services such as *LocationManager* and *Audio Services*. The end result is that the applications that form part of the device conforms to these finer-grained security policies rather than utilizing the ones the device provides. A final effort [41] involves modifying Android OS and applies context-based access control restrictions in mobile devices. The intent is to allow a user of a mobile device to create a security policy that establishes which resources/services of his/her mobile device their installed mobile applications should have access to. In contrast to these efforts, our work presented in this section focuses on applying access control mechanisms to the APIs that are not part of the mobile system itself. In addition, most of these works are specific for Android OS/API while ours can be implemented for any type of application (even though we focus on the mobile setting) since our access control approach is enforced server-side.

The third area of related work involves role-based access control and extensions that expand RBAC with context-aware techniques in order to provide finer-grained access control security policies to those systems that contain highly sensitive data. One effort does this by proposing an RBAC model with a spatiotemporal extension for web applications [42] and another effort proposes a similar approach but for mobile applications [43]. The access control system made for web applications [42] can be applied to an existing system as a dll component. Another approach proposes a dynamic RBAC approach for Android devices [44]. That approach focuses on modifying the Android framework to provide a uniform security policy to mitigate security risks in mobile devices that are utilized by users who are part of an enterprise. Moreover, an effort [45] proposed a model that extends RBAC to generate RBAC conceptual policies. Nevertheless, the aforementioned effort does not provide details of which specific application domain(s) the approach could support. Finally, the approach [46] utilizes user attributes to provide access control for business processes in mobile computing and uses RBAC in combination of context-aware access control mechanisms. Basically, the approach identifies the tasks that are available in a system, assigns roles to the users of that system and, establishes which roles have permissions of which tasks and under what context. Our approach can augment their work by including additional RBAC, MAC, and DAC capabilities server-side. Our Intercepting API Calls approach could easily be extended to support other types of access control such as attribute-based access control (ABAC) [47] and identity-based access control (IBAC) [48] in order to generate finer-grained access control. Our Intercepting API Calls approach can also be applied to mobile web applications and, it is not domain-specific; this contrasts to the discussed related work.

9 Conclusion

This paper presented an API-based access control approach on the interactions between the UI and the mobile applications' API services to control by RBAC, MAC, and/or DAC permissions which invocations are allowed to be invoked on a user-by-user basis through the generation of an intercepting API that mirrors the original mobile application's API. Section 1 introduced what we wanted to accomplish with the

Intercepting API Calls approach. Section 2 defined background concepts, motivated the Intercepting API Calls approach by explaining the important role of the API in accessing information, and exposed details of the CT² mHealth app. Section 3 presented the high-level processing of the Intercepting API Calls approach using the classic architecture of the User Layer, Presentation Layer, Business Layer, and, Data Layer. Section 4 defined a service-based model for RBAC, MAC, and DAC, and detailed the underlying processing of the Intercepting API Calls approach by examining the way that API services are categorized. Section 5 examined the RBAC/MAC/DAC approach for Intercepting API Calls with a focus on the Interactions and Infrastructure for the Intercepting API Calls approach. Section 6 explored the automatic generation of RBAC, MAC, and DAC enforcement code for the Intercepting API Calls approach via an algorithm. Section 7 illustrated the Intercepting API Calls approach via the CT² mHealth application. Finally, Sect. 8 discussed related work in security and access control mechanisms for mobile applications.

References

1. Heisey-Grove, D., Patel, V.: Any, certified, and basic: quantifying Physician EHR adoption through 2014, September 2015. https://www.healthit.gov/sites/default/files/briefs/oncdatabrief28_certified_vs_basic.pdf
2. Walker, J., Pan, E., Johnston, D., Adler-Milstein, J., Bates, D.W., Middleton, B.: The value of health care information exchange and interoperability. *Health Aff.* **24**(2), 10–18 (2005)
3. Himss: Meaningful use stage 3 final rule. <http://www.himss.org/ResourceLibrary/genResourceDetailPDF.aspx?ItemNumber=44987>. Accessed 2016
4. Rindfleisch, T.C.: Privacy, information technology, and health care. *Commun. ACM* **40**(8), 93–100 (1997)
5. Sandhu, R.S., Samarati, P.: Access control: principles and practice. *IEEE Commun. Mag.* **32**(9), 40–48 (1994)
6. Ferraiolo, D., Kuhn, R: Role-based access control. In: Proceedings of the NIST-NSA National (USA) Computer Security Conference, pp. 554–563 (1992)
7. Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, D., Chandramouli, R.: Proposed NIST standard for role-based access control. *ACM Trans. Inf. Syst. Secur. (TISSEC)* **4**, 224–274 (2001)
8. Bell, D.E., La Padula, L.J.: Secure computer system: unified exposition and multics interpretation. MITRE Corp. (1976)
9. Department of Defense: Department of Defense Trusted Computer System Evaluation Criteria, 26 December 1985
10. West, A.: 5 Roles of role based access control (2015). <https://www.itouchvision.com/5-roles-of-role-based-access-control-the-software-security-guard/>
11. Fernández-Alemán, J., Señor, I., Lozoya, P., Toval, A.: Security and privacy in electronic health records: a systematic literature review. *J. Biomed. Inform.* **46**(3), 541–562 (2013)
12. FHIR: Welcome to FHIR (2016). <https://www.hl7.org/fhir/index.html>
13. Cobb, M.: API security: how to ensure secure API use in the enterprise. <http://searchsecurity.techtarget.com/tip/API-security-How-to-ensure-secure-API-use-in-the-enterprise>. Accessed 11 Mar 2014

14. Rivera Sánchez, Y.K., Demurjian, S.A.: Towards user authentication requirements for mobile computing. In: Malik, A., Anjum, A., Raza, B. (eds.) *Innovative Solutions for Access Control Management*, pp. 160–196. IGI Global (2016). <http://www.igi-global.com/book/innovative-solutions-access-control-management/146981>
15. Rivera Sánchez, Y.K., Demurjian, S.A., Conover, J., Agresta, T., Shao, X., Diamond, M.: An approach for role-based access control in mobile applications. In: Mukherja, S. (ed.) *Mobile Application Development, Usability, and Security*, pp. 117–141. IGI Global (2016). <http://www.igi-global.com/book/mobile-application-development-usability-security/154083>
16. Rivera Sánchez, Y.K., Demurjian, S.A., Gnirke, L.: An intercepting API-based access control approach for mobile applications. In: *Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST)*, pp. 137–148. April 2017. <http://www.scitepress.org/Papers/2017/63543/63543.pdf>
17. Rivera Sánchez, Y.K., Demurjian, S.A., Baihan, M.: Achieving RBAC on RESTful APIs for mobile apps using FHIR. In: *Proceedings of the 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pp. 177–184. IEEE Mobile Cloud, April 2017. <https://ieeexplore.ieee.org/document/7944884/>
18. Beal, V.: API - application program interface (2004). <http://www.webopedia.com/TERM/A/API.html>
19. Facebook (2014). <https://developers.facebook.com/docs/graph-api>
20. Flanders, D., Ramsey, M., McGregor, A.: The advantage of APIs (2012). <https://www.jisc.ac.uk/guides/the-advantage-of-apis>
21. REST API Tutorial, Learn REST: A RESTful Tutorial (2012). <http://www.restapitutorial.com/>
22. Rouse, M.: HTTP (Hypertext Transfer Protocol), 15 July 2006. <http://searchwindevelopment.techtarget.com/definition/HTTP>
23. Collet, S.: API security leaves apps vulnerable: 5 ways to plug the leaks (2015). <http://www.csoonline.com/article/2956367/mobilesecurity/api-security-leaves-apps-vulnerable-5-waysto-plug-the-leaks.html>
24. Snapchat (2011). <https://www.snapchat.com/>
25. Snapchat: Finding Friends with Phone Numbers (2013). <http://blog.snapchat.com/post/71353347590/finding-friends-with-phone-numbers>
26. Zeman, E.: Snapchat lays down the law on third-party apps (2015). <http://www.programmableweb.com/news/snapchat-lays-down-law-third-partyapps/2015/04/07>
27. Instagram (2010). <https://www.instagram.com/>
28. Dellinger, A.J.: This Instagram app may have stolen over 500,000 usernames and passwords (2015). <http://www.dailydot.com/debug/instaagent-instagram-app-malware-ios-android/>
29. Larson, S.: Instagram restricts API following password breach, will review all apps going forward (2015). <http://www.dailydot.com/debug/instagram-api-restrictions/>
30. Connecticut General Assembly: Substitute for Raised H.B. No. 6722 (2015). https://www.cga.ct.gov/asp/CGABillStatus/CGABillstatus.asp?which_year=2015&selBillType=Bill&bill_num=HB6722
31. Microsoft Corporation: *Mobile Application Architecture Guide* (2008). http://robtiffany.com/wpcontent/uploads/2012/08/Mobile_Architecture_Guide_v1.1.pdf
32. Cappos, J., Wang, R., Yang, Y., Zhuang, Y.: Blursense: dynamic fine-grained access control for smartphone privacy. In: *Sensors Applications Symposium (SAS) 2014*. IEEE (2014). <https://doi.org/10.1109/SAS.2014.6798970>
33. Xu, Z., Zhu, S.: Semadroid: a privacy-aware sensor management framework for smartphones. In: *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pp. 61–72. ACM (2015)

34. Beresford, A., Rice, A., Skehin, N., Sohan, R.: MockDroid: trading privacy for application functionality on smartphones. In: 12th Workshop on Mobile Computing Systems and Applications, Phoenix, Arizona (2011)
35. Benats, G., Bandara, A., Yu, Y., Colin, J., Nuseibeh, B.: PrimAndroid: privacy policy modelling and analysis for android applications. In: Symposium on Policies for Distributed Systems and Networks (POLICY 2011) (2011)
36. Wang, Y., Hariharan, S., Zhao, C., Liu, J., Du, W.: Compac: enforce component-level access control in android. In: Proceedings of the 4th ACM Conference on Data and Application Security and Privacy, San Antonio, Texas, USA (2014)
37. Jin, X., Wang, L., Luo, T., Du, W.: Fine-grained access control for HTML5-based mobile applications in android. In: 16th Information Security Conference (ISC) (2015)
38. Hao, H., Singh, V., Du, W.: On the effectiveness of API-level access control using bytecode rewriting in android. In: 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, Hangzhou, China (2013)
39. Backes, M., Bugiel, S., Gerling, S., von Styp-Rekowski, P.: Android security framework: extensible multi-layered access control on android. In: 30th Annual Computer Security Applications Conference (2014)
40. Bugiel, S., Heuser, S., Sadeghi, A.: Flexible and fine-grained mandatory access control on android for diverse security and privacy policies. In: 22nd USENIX Security Symposium, pp. 131–146 (2013)
41. Shebaro, B., Oluwatimi, O., Bertino, E.: Context-based access control systems for mobile devices. Fellow, IEEE (2015)
42. Aich, S. Mondal, S., Sural, S., Majumdar, A.K.: Role based access control with spatiotemporal context for mobile applications. *Trans. Comput. Sci.* **IV** (2009). Special Issue on Security in Computing
43. Abdunabi, R., Sun, W., Ray, I.: Enforcing spatio-temporal access control in mobile applications. *Computing* **96**(4), 313–353 (2014)
44. Rohrer, F., Zhang, Y., Chitkushev, L., Zlateva, T.: DR BACA: dynamic role based access control for android. In: 29th Annual Computer Security Applications Conference, New Orleans, Louisiana, USA (2013)
45. Fadhel, A., Bianculli, D., Briand, L., Hourte, B.: A model-driven approach to representing and checking RBAC contextual policies. In: CODASPY 2016, pp. 243–253. ACM (2016)
46. Schefer-Wenzl, S., Strembeck, M.: Modelling context-aware RBAC models for mobile business processes. *Int. J. Wireless Mob. Comput. (IJWMC)* **6**(5), 448 (2013)
47. Coyne, E., Weil, T.: ABAC and RBAC: scalable, flexible, and auditable access management. *IT Prof.* **15**(3), 14–16 (2013)
48. Jericho Systems (2014). https://www.jerichosystems.com/technology/glossaryterms/identity_based_access_control.html



Assisted End User Development for Non-programmers: Awareness, Exploration and Explanation of Composite Web Application Functionality

Carsten Radeck^(✉) and Klaus Meißner

Faculty of Computer Science, Technische Universität Dresden,
Dresden, Germany
{carsten.radeck,klaus.meissner}@tu-dresden.de

Abstract. Mashing up existing components allows end users to build custom web applications in order to fulfill situation-specific needs. However, creating and even using mashup applications still turn out to be complicated tasks for non-programmers. Main challenges include understanding the composite nature of mashups and their functionality. Non-programmers typically lack awareness for inter-widget communication and understanding of the functionality individual components and especially a set of combined components provide. Thus, they may struggle to use components as intended. Prevalent mashup approaches provide no or limited concepts and assistance tools in this regard, resulting in more or less successful trial and error strategies of users. In this paper, we introduce novel techniques for exploration, explanation and awareness of mashup functionality. These concepts assist non-programmers to better understand and to be aware of the capabilities that components and their interplay in a mashup provide. Based on annotated component descriptions, interactive explanations and stepwise tutorials are generated and presented directly in the components' user interface. Additionally, mashup functionality can be explored and active inter-widget communication is visualized to foster awareness of non-programmers. We present our iterative design process which led us from early concepts towards our current solution. The proposed approach is validated with the help of a prototypical implementation within our mashup platform and evaluated by means of a user study. The results indicate that our concepts help non-programmers to better understand and leverage the functionality of composite web applications.

Keywords: Mashup · Awareness · Inter-widget communication
Generated tutorials · End User Development · Assistance

1 Introduction

As the number of web resources, services and application programming interfaces increases, the mashup paradigm is an approach to re-use and combine such com-

ponents in a broad variety of application scenarios to provide added value. Universal composition approaches enable platform-independent modeling of mashups. Therein, apply components spanning all application layers, ranging from data and logic services to user interface (UI) widgets, are uniformly described and composed.

In principle, the mashup paradigm and end user development complement each other quite well, allowing to better meet situational niche requirements of users. However, it is still cumbersome for end users, and especially non-programmers, to develop and even use composite web application (CWA). They face several challenging tasks in CWA development and usage, as for instance pointed out by [1, 2]. In this paper, we pay special attention to the following two challenges: (1) understand what single components are capable of and what functionality they provide in interplay, as well as (2) become aware of inter-widget communication (IWC).

Our platform adheres to universal composition and enables users to build and use custom, situational CWA. Non-programmers can modify an application while it is executed. This way, they get instant feedback on their composition actions. Components are semantically annotated with the capabilities they provide. Building up on this, capabilities of arbitrary composition fragments are automatically calculated [1]. This allows our mashup environment to offer tools for continuous assistance. With this, we tackle the challenges that we mentioned above and that we detail in a **reference scenario** now.

Non-programmer Bob uses an existing mashup for travel planning recommended by a friend. It combines two maps, a route calculator, a weather widget and two components for searching points of interest and hotels. Since Bob is neither familiar with the overall mashup nor the individual components, he has understanding problems of which functionality the application provides and which not and how to achieve it. For example, Bob is unsure why there are two maps, whether the location in a map effects other components, and if yes, which kind of effect. Further, he does not know how to find hotels near the target location. Normally he would have to explore the application manually in a trial and error style, but platform supports Bob in gaining insight. First, an overview panel displays the mashup functionality, possibly composed of several sub-functionalities. It allows Bob to inspect what tasks he can solve with the application at hand and which components partake in certain functionalities. This way, Bob comprehends that one map serves for selecting the start location, while the other is used to select the target location of a route. Since Bob has no clue about how to see a list of routes, he activates an explanation tool, which shows interactive animations of necessary steps and interactions he has to perform. It highlights elements of the component UI, indicates sequential and parallel work-flows and complements this with textual descriptions of what to do. Such explanations not only work for component interplay, but for capabilities of single components and how these are reflected on the component UI, too. This helps Bob understand that he can move a marker or type the location name in an input field of either map to select a location. Bob has not yet noticed the communication relation between one map and the weather widget, since these are positioned far away from each other on the screen. A platform feature animates

the data flow whenever it occurs, utilizing the same visualization techniques as in the explanation tool. Consequently, Bob gets aware of data transfer between both widgets.

In order to implement the reference scenario and to tackle the challenges stated above, there are at least the following foundational **requirements**:

- There need to be appropriate mechanisms that explain mashup functionalities to non-programmers. Tutorials for components can be statically provided by developers, but this is not feasible for CWA since they are developed ad-hoc by end users combining components in unforeseen ways. Explanations should be optically consistent for all CWA. Hence, it seems inappropriate to rely on component-provided tutorials and combine them. Explanations have to be offered automatically for arbitrary mashups, should be interactive and directly presented in the UI of components.
- Concepts are required to foster awareness for IWC when it occurs. An approach has to work for arbitrary communication relations and should directly operate in components' UI, since this actually can help users, as pointed out by [3].

Most prevalent mashup approaches support users with visual composition metaphors and recommendations. But assisting users in understanding a CWA and in becoming aware of IWC are neglected or very limited so far. Traditional approaches like static help pages and forums lack suitability, too. As a main **contribution**, we introduce a set of generic concepts for exploration, explanation, and awareness of the capabilities of arbitrary CWA in this paper. These techniques allow non-programmers to investigate the functionality of single components and the entire mashup. With the help of component annotations, step-wise tutorials that give instructions on how to achieve capabilities are generated. Thereby, textual explanations and graphical highlights in the component UI are combined. In addition, we propose concepts to make users aware of IWC when it occurs. These techniques are especially valuable as they work for unforeseen combinations of black-box components. We conduct a user study to evaluate our concepts.

This article is an extended version of the paper presented at WEBIST 2017 [4]. The remaining paper is structured as follows. In Sect. 2 we discuss related work. We outline our overall assisted end user development (EUD) approach in Sect. 3 then. First iterations of our design process are subject of Sect. 4. Based on the lessons learned, we introduce our concepts for exploration, explanation and awareness of IWC in Sect. 5. Details on our implementation and on a user study we conducted are described in Sect. 6. Finally, Sect. 7 concludes the paper and outlines future work.

2 Related Work

Three types of performance support can be distinguished [5]. *Intrinsic support* is inherent to the system, seamlessly integrated with the UI as well as the behavior of the system. There is no break in users' workflow, but such techniques

are limited in extent because they compete with the normal application for the limited UI space. *Extrinsic support* is directly integrated with the system as well, yet not in the primary workspace. Such support is often contextualized regarding the user task and typically requires the user to invoke or accept it [5]. Examples include explanations, demonstrations and wizards. Our approach can be assigned to this category. *External support*, like forums, tutorials and help pages on the web, is not integrated with the workspace itself. Since it typically lacks context specificity, users must ad hoc transfer provided knowledge to their current task context. External support is well suited for developer-provided content, like components, and introductions to static content. However, in context of CWA, that serve short-termed, situation-specific needs, combine components in unforeseen ways, and are subject iterative refinement, they lack practicability and suitability.

A goal of the *Idea Garden* [6] is to help users to overcome design barriers. To this end, suggestions for problem solving strategies are presented in an extrinsic manner. Suggestions also include step-by-step instructions on how to perform a strategy. However, suggestions are often general and context-free and establish no direct link to the component UI. The *Whyline* [7] provides support for users to debug their programs. To this end, developers construct “why did” and “why did not” questions using menus and referencing objects involved in an algorithm. An answer is shown as a graph, visualizing actions that happened at runtime and arrows that represent causality. This approach operates on a more source code oriented level than our concept and there is no visualization of active data transfer. A debugging interface for Yahoo! Pipes is proposed in [8]. Certain types of anomalies are automatically detected in pipes. Users are supported by a “to-fix list” of bugs that highlights corresponding erroneous modules, displays error messages and gives hints on possible solutions. This approach helps users to identify problems that hinder a pipe from working properly, while we try to explain which functionality a working mashup provides and how a user has to interact with it.

Most mashup EUD platforms provide at least external support. But as we explicate in the remainder of this section, intrinsic and extrinsic approaches are missing or very limited. There are composition metaphors that can also be used to explain what should go on in a mashup. Typically, a derivation of “if-then” is employed, e. g., in *IFTTT*¹, *CapView* [9], *PEUDOM* [10] and *NaturalMash* [11]. In rather simple scenarios, the resulting composition logic is suitable to explain functionality to non-programmers. However, the required link to the actual UI is missing in *IFTTT* and *CapView*. In the *PEUDOM* platform communication channels are created and modified in a dedicated dialog by selecting an event and an operation. To explain a channel, a short sentence describes the causal relation. However, there is no means for explaining the overall mashup functionality. In a widget editor, the link between data schema and UI can be established graphically. But these techniques are not utilized to explain widget functionality later on. In *NaturalMash* [11] restricted natural language defines

¹ <https://ifttt.com/>.

the composition logic. Conceptually, when selecting text fragments, corresponding UI elements are highlighted. However, neither details nor an evaluation on this aspect are available. Similar to this work, we utilize generated labels, but also present animations for complex composition fragments directly within the UI of a CWA. In the *FAST-Wirecloud* approach [12] *behaviors* are composition fragments of a mashup that represent a functionality. Behaviors feature natural language descriptions and are used to ease the composition process, but not to explain a mashup at runtime. SOA4all [13] propose assisted composition of web services. Therein, templates are prototypical compositions and show similarity to task models. Users can choose templates from a taxonomy and have to select a service per task in order to complete the composition. The association of service to task is a high-level explanation of a service's functionality. However, besides that, there are no other features. Several approaches like [14–16] display the composition logic including inter-component connections with the help of abstract graph visualizations. However, such concepts are rather technical and presenting the actual functionality achieved and its relation to the UI is not considered. [14] additionally support users with a generated textual description of the mashup composition at hand. Again, this text is rather technical than domain-oriented. Further there is no relation to the component UI and no means to make users aware of IWC at runtime.

The *OMELETTE* approach features a live development mashup environment. A distinct composition paradigm is followed, where per default all possible communication channels between widgets are established. As pointed out in [2], this results in challenges for awareness and control of users. Thus, features fostering awareness for IWC have been proposed [3]. The communication relations of widgets are visualized as arrows on the canvas and active data transfer is indicated by highlighting involved widgets. However, the authors point out, that suitable visualization techniques are still an open research question. Further, explanations of the functionality communication relations provide as well as tutorials are missing. Within the Linked Widgets Platform [16] active widgets are highlighted, which can be considered as a limited means to make users aware of IWC. The *MashupEditor* [17] highlights UI elements of widgets and draws arrows to illustrate communication channels. We use similar base techniques, but also present animations and textual explanations to the user.

To sum up, none of the presented concepts fulfills all the requirements stated in Sect. 1. Therefore, we propose novel, generic techniques for exploration, explanation and awareness of CWA functionality, that are adequate for non-programmers.

3 Assisted End User Development of Mashups with CRUISE

In this section we outline the core characteristics and foundations of the CRUISE mashup platform [1] and relate the concepts we present in this paper to it.

The CRUISE mashup platform strives to enable end users to build custom CWA. Thereby, we specifically address *non-programmers*, defined as web-experienced domain experts without programming knowledge. They typically know their problem and possible solutions in terms of domain concepts and tasks to perform, but fail to map such solutions on technical requirements or mashup compositions. Composition platforms for non-programmers should meet crucial requirements, see e. g. [12,13]. Considering this, applying scenario-based techniques and conducting user studies, we derived the following design goals for our approach.

Development Process. An iterative process with continuous assistance and instant feedback on composition steps is required.

Level of Abstraction. Technical details, concepts and terminology have to be hidden from users. Instead, when communicating with non-programmers, domain concepts and vocabulary should be utilized.

Assisted Start. Appropriate tools for accessing or searching components and applications are necessary, especially in case a user has only a vague idea.

Assisted Composition. An intuitive composition metaphor and automation are required.

Recommender System. Users should be supported by recommendations during development. In context of this paper it is especially relevant, that recommendations are presented in a suitable way.

Assist Usage and Understanding. As pointed out in Sect.1, non-programmers need support to explore, understand and be aware of the composite functionality of CWA.

In order to fulfill these requirements, as shown in Fig. 1, our EUD approach builds up on a semantic model layer, which is the foundation for several base mechanisms. Both layers enable EUD tools, suitable for non-programmers, according to our development and usage paradigm, which we denote *Live Sophistication*.

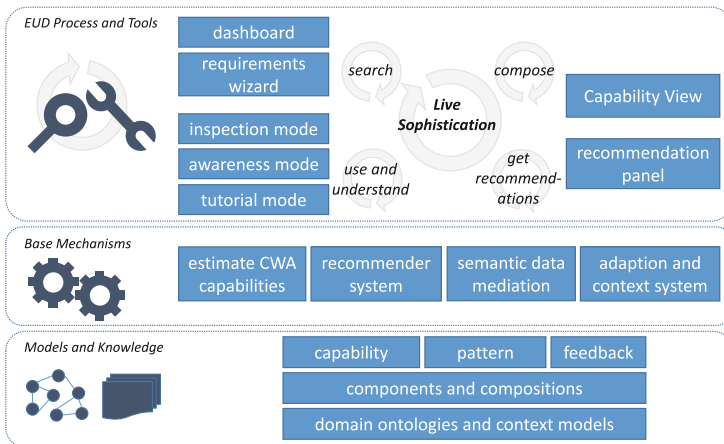


Fig. 1. Overview of the CRUISE platform for mashup EUD.

3.1 Model Layer

The CRUISE platform follows a model-driven approach to create and execute mashups. Arbitrary Web resources and services can be encapsulated as components. In line with universal composition, components of all application layers are uniformly described and treated as black boxes, based on a common, generic *component model*. The latter characterizes components by means of several abstractions: events and operations with typed parameters, typed properties, and capabilities. The declarative Semantic Mashup Component Description Language (SMCDL) provides a concrete syntax for the component model. In addition, it features semantic annotations to clarify the meaning of component interfaces and capabilities [9]. To this end, properties and parameters refer to domain ontology concepts to indicate data semantics. All aspects of a CWA are modeled with the help of the *Mashup Composition Model (MCM)*. This includes the components to be integrated, application screens with their layout and transitions, and the event-based communication of components.

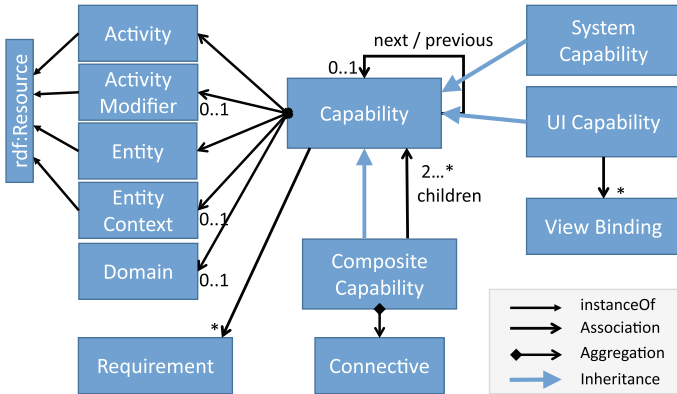


Fig. 2. The capability meta-model [4].

Capabilities allow to model functional and behavioral semantics of composition fragments, i.e., of components, applications as well as patterns. Inspired by research on task models, capabilities describe what a composition fragment is able to do or which functionality it provides, like displaying a location or searching hotels. To this end, capabilities essentially are tuples (*activity, entity*) – denoted **activity entity** from now on – and express which activity or task is performed on or with which domain object, e.g. **search hotel**. As shown in Fig. 2, references to semantic concepts described in ontologies back the description with formal semantics providing domain-specific knowledge and allow for reasoning. To achieve a capability, it may be necessary for the user to partake and interact with the UI or not. Therefore, UI and system capabilities are distinguished. To establish hierarchical structures, capabilities can be *composite*.

The relation of children of a composite capability is defined by a *connective*, e. g., parallel and sequential. In case of sequences, capabilities are chained to define the order using *next* and *previous*. See Fig. 3 for an example of a composite capability.

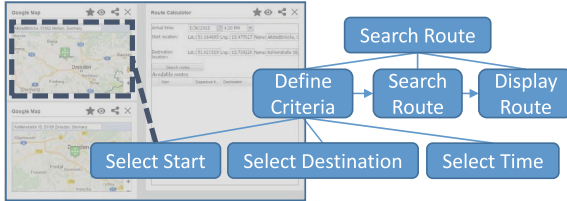


Fig. 3. Exemplified composite capability with an indicated view binding.

View bindings are a concept particular for UI capabilities. They link the semantic layer and the UI of a component, as indicated in Fig. 3. Basically, a view binding comprises interaction steps, modeled as atomic, parallel or sequential operations. Atomic operations point to DOM elements, using a selector language, e. g. CSS selectors, can name the elements and define interaction techniques, like click and sweep (see lines 3 and 6 in Listing 1). To reduce complexity of annotating and algorithmic processing, composite operations are restricted to one layer of atomic operations. Each UI capability can be equipped with multiple view bindings, which are considered alternatives then, e. g., if it is possible to select a location via typing something in a text field or double clicking a map (lines 2 and 5).

```

1 <capability activity="Select" entity="Location">
2   <viewbinding>
3     <atomicoperation element="input[id$='mapTextField']" interaction="type"
4     />
5   </viewbinding>
6   <viewbinding>
7     <atomicoperation element="div[id$='locationMarker']" elementName="map
8     marker" interaction="dragAndDrop" />
9   </viewbinding>
10 </capability> ...

```

Listing 1. Excerpt of a SMCDDL descriptor (from [4]).

In CRUISE, *composition knowledge* is represented by multiple, complementing aspects. *Patterns* model meaningful or statistically significant composition fragments, e. g., complex constellations of several components connected via certain channels. In addition, *domain ontologies* are leveraged to derive more abstract patterns on a semantic level and to estimate the semantic relevance of patterns in a given context. Finally, implicit and explicit user *feedback* on components, CWA, and recommendations enriches those aspects. Since the feedback

context, comprising information like the user, composition context, and recommendation type, is kept, validity and suitability of composition fragments and recommendations in certain contexts are learned over time.

3.2 Base Mechanisms

Assisted EUD of mashups comes with a set of base algorithms and mechanism, which build up on the model layer and offer the functionality required by EUD tools.

Semantic data mediation techniques [18] foster component re-use by solving interface incompatibilities, increasing the opportunities to combine components. For instance, different granularity, abstraction and units of component interface parameters can be mediated as long as the underlying domain models allows that.

Based on capabilities of components and communication channels within a mashup, an algorithm estimates the capabilities of patterns and applications [1]. The concepts we present in this paper build up on the derived capability models.

A *recommender system* gives advice on composition fragments to users [19]. In order to embrace the highly iterative and ad-hoc development of CWA and to meet the requirements of our target group of users, *triggers* define when recommendations should be presented. Several trigger types can be distinguished, e. g., user-demanded, proactive and reactive. This allows our platform to cover a multitude of scenarios and conditions, providing assistance throughout the development and usage process. Deriving recommendations is based on *patterns* and semantic knowledge. *Viewers* are responsible for displaying recommendations to users. Implicit and explicit feedback, for example, selection statistics and user ratings, is gathered by the recommender system and influences candidate ranking. Accepted recommendations are automatically integrated into a new or the current CWA. To this end, patterns are mapped to adaptation techniques and the adaptation system implements those techniques.

3.3 EUD Process and Tools

A fundamental feature of our approach is the mashup development and usage paradigm, which we refer to as *Live Sophistication*. Therein, run time and development time of a CWA are strongly interwoven. Users can seemingly switch between editing and using an application. Communication with users takes place on capability level and necessary mappings to composition model changes are handled transparently by the platform. According to *Live Sophistication*, non-programmers perform several activities in a highly iterative manner: *search* for suitable composition fragments; *compose* an application, e. g., add components and connect them; users *get recommendations*, either explicitly on demand or implicitly upon system initiative; they *use* a CWA and have to *understand* its purpose and how to handle it. As indicated in Fig. 1, most activities are optional and iterative in themselves. Further, appropriate EUD tools support each activity.

In order to realize Live Sophistication, the CRUISE architecture, illustrated in Fig. 4, features a mashup runtime environment (MRE) that not only manages mashups’ lifecycle, but additionally provides most base mechanisms and a set of EUD tools. An MRE relies on server-side repositories and services.

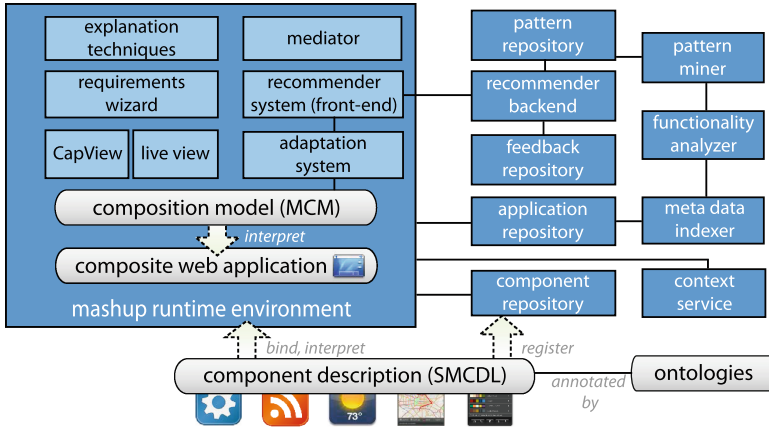


Fig. 4. Architectural overview of the CRUISE platform.

As an important assistance feature, our platform helps non-programmers in getting started with their mashup composition. A personalized dashboard offers several entry points. There are listings of recently used and otherwise relevant components and compositions, enabling users to browse the catalog. Furthermore, multiple search options are offered, like keyword-based and faceted search. A *wizard* guides non-programmers in expressing and decomposing their functional requirements in terms of capabilities.

In addition, a MRE is equipped with composition tools. For instance, different views on the current CWA are provided: The live view is mainly intended for usage and thus only component UIs are visible to the user. Complementing this, there are overlay views, like the CapView [9], that display component and composition model details and mainly serve for development purposes. Furthermore, tools like the aforementioned CapView and our novel explanation techniques, which are subject to this paper, explain the capabilities originating from components and their interplay within the current mashup in a textual and visual manner. Regarding CapView, we re-use its label generator, which creates human readable text from capabilities, but provide explanation techniques directly in the component UI rather than in an abstract view.

Recommendations are shown to users in dedicated viewers, e.g., the recommendation panel, and viewers integrated in other platform utilities, like CapView. Viewers and rating dialogs are responsible to collect user feedback, which is stored in conjunction with the feedback context.

The *explanation techniques* we present in this paper, serve to assist non-programmers in exploring and understanding the capabilities of CWA.

3.4 Summary

In summary, referring to the high-level design goals stated at the beginning of this section, we propose the following solutions and approaches.

Development Process. In Live Sophistication, development and run time of a CWA are seemingly interwoven. This allows for highly iterative processes and fosters instant feedback on user’s composition actions. An MRE not only executes mashups but is equipped with proper tools, support each phase.

Level of Abstraction. Capabilities, respectively, their view bindings and the labels from their annotations, are the main means to communicate with users. This way, domain tasks and entities are emphasized while technical concepts are completely hidden. We adhere to the WYSIWYG principle in our MRE and all tools where applicable.

Assisted Start. Our platform provides several entry points into the composition and usage process of CWA. Dashboard as well as requirements wizard enable exploratory search, addressing users with vague ideas.

Assisted Composition. Utilizing domain ontologies, pattern mining, and semantic data mediation, CRUISE is able to derive composition knowledge and automatically integrate it. From a users point of view, composition takes place at capability level.

Recommender System. Our platform features a hybrid recommender system that suggests components, applications as well as patterns. Recommended patterns are visualized using capabilities. We detail our solution in Sect. 5.3.

Assist using and Understand. Our approach comprises novel *explanation techniques* that help non-programmers to understand and utilize composite applications.

In this paper we specifically focus on our proposal for assisting users to understand, explore and be aware of the capabilities a CWA provides. Next, we describe first iterations of our design process and the lessons learned.

4 First Approaches and the Lessons Learned

Although the primary purpose of the **CapView** [9] is performing composition tasks, it also features basic means for inspection and navigation of component capabilities. CapView is an overlay of the live view and positions capability representations above the components that provide them (see Fig. 5). Connection lines between capabilities represent communication channels on capability level. Labels are generated according to a rule set, incorporating ontology concepts referred to in capabilities. When a capability is selected, the labels of all other capabilities, with which it can be connected, are adapted to create short

sentences which reflect cause and effect. This way, the functionality of communication channels is explained to users. Non-UI components are handled specially. In CapView, a representative icon is attached to connections where the component is involved. The live view features a **non-UI component panel**, listing components by name and icon. In both cases, generated labels state the functionality.



Fig. 5. CapView prototype in a rather complex CWA [4].

We evaluated this first iteration of concepts with several small user studies. As a main result of the study presented in [9], we found that task-oriented labels, their contextualized adaptation to form sentences and positioning capabilities above components has potential. But we also identified some drawbacks. For example, since CapView is an overlay and abstracts from instance data and component UI, some users struggled to understand how to actually achieve capabilities. We conducted a further small user study with the aim to deepen our problem understanding by identifying conceptual and usability issues users face in our composition environment. Five non-programmers participated, were briefly introduced to our prototype and then asked to solve several tasks. Different CWA, like a route search mashup with two maps and a route calculator, were presented to the users. The tasks aimed at testing if users understand component and application functionality and if existing tools are accepted and usable. As example tasks, the participants were asked to identify the overall mashup capabilities and to name components contributing to capability **search route**. They had to list which components interact and entitle the resulting functionality, and were asked to describe how to **search points of interest**. Finally, we encouraged the participants to freely use a CWA and thereby mention understanding problems and suggest improvements.

This user study delivered manifold results. We confirmed some known and identified new issues with CapView as a tool to understand CWA. As main problems we identified switching into a dedicated view, i. e., leaving the live view, and

users struggling with realizing the link between CapView and corresponding elements in the component UI. In consequence, users experimented in the live view rather than using CapView. Few users were unable to cope with CapView at all. In general, we found that even in rather simple scenarios users may face understanding problems when using components or applications they are unfamiliar with or which do not match with their expectations. For example, some users misinterpreted the purpose components serve for, since they expected that it is determined by positioning on the canvas or the temporal sequence they interact with components. Moreover, users were sometimes unaware of inter-widget communication, e. g., between map and weather widget. In this point, we confirm the findings of [2]. Directed data flow caused confusion when a bidirectional synchronization was assumed, e. g., between maps and route calculator. Users often ignored the non-UI component panel or misunderstood it as advertisement.

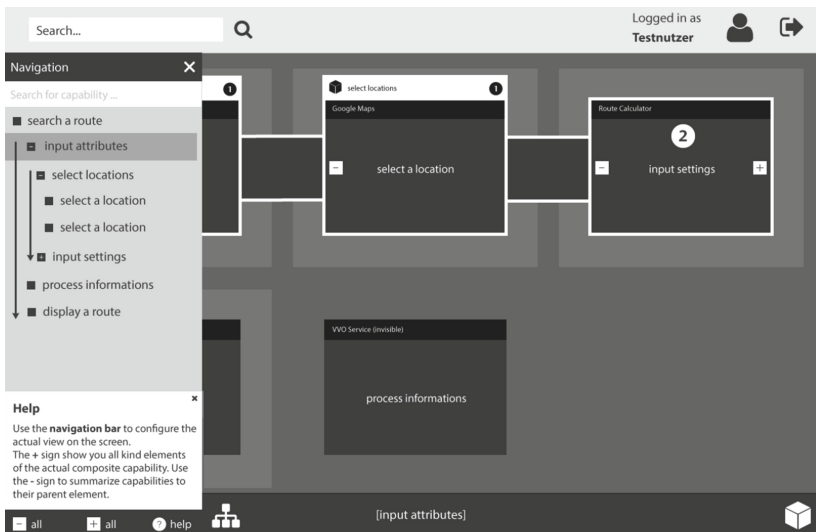


Fig. 6. The discontinued CompCapView approach [4].

Since CapView is not designed to give hints about overall capabilities of a CWA at hand, we developed **CompCapView** as an extension that supports composite capabilities, see Fig. 6. Basic principles are the same, e. g., it overlays the live view and capability representations are drawn above the components that provide these capabilities. Depending on the current position within the capability hierarchy, representations of child capabilities are drawn and connected to each other. In case of a sequential relation of capabilities, their order is mentioned by numbers. All root capabilities are shown initially. CompCapView allowed to zoom in and out within the hierarchy. Additionally, a **panel displaying CWA capabilities** as a tree-like structure was developed. It is synchronized with the current view. In case of CompCapView it propagates the

currently selected capability which is focused and highlighted subsequently. In context of the live view, all components providing or partaking in the selected capability are highlighted.

For evaluation purpose, we conducted a small user study according to the thinking aloud method. Six persons participated, which were in average 23 years old and all had no knowledge about mashups and programming. They worked or studied in different sectors, like medical engineering, retail, economy, and social pedagogy. The study based on a prototype of CompCapView and the capability panel within our MRE.

After a brief introduction to the concepts of CompCapView, participants were asked to solve several tasks that tested the core principles of CompCapView. For instance, the participants should name the overall capability of a given application and describe the sequence in which components cooperate to achieve it. In addition, we were for instance interested in whether the navigation between layers is comprehensible and usable and if the participants understand the meaning of connections. Finally, we measured the SUS and Task Load Index. As a main result, all of the participants were able to determine the overall mashup capability and 5/6 stated correctly which components actually contribute to mashup functionality. Most participants understood the layering concept (4/6) and the relation of composite capability representations and the actual mashup components (5/6). But only 3 of 6 participants correctly identified sequential or parallel capabilities. Several remarks and comments addressed implementation rather than conceptual issues. However, as a crucial conceptual problem connections of composite capabilities were misinterpreted as an indicator for order, which was further complicated by assumed reading directions. This study revealed, that especially the navigation panel was considered useful since it provides an overview and quick access to CWA capabilities. The resulting average system usability scale (SUS) score of 66 indicates acceptable, yet limited usability. The Task Load Index questionnaires attested medium mental demand and effort. Although the participants considered frustration low to medium and time demand low, and were satisfied with their performance, this underpins that the ComCapView concepts have some potential but lack suitability for non-programmers.

In summary, from these findings and observations, we amongst others conclude that:

- explanations techniques should operate directly within the UI of components;
- visual capability connections have to carry an intuitive semantics; and
- composite capabilities should be accessible via an overview panel.

Next, we present our current approach that incorporates the lessons learned.

5 Current Approach: Explanation Techniques for CWA

In this section, we present in detail our concepts for assisting users to understand what functionality a CWA provides and how to use it. The proposed explanation techniques serve different purposes, like to give an overview on capabilities, provide detailed stepwise instructions and make users aware of IWC.

5.1 Preliminaries and Foundation

As already mentioned in Sect. 3, components are semantically annotated with capabilities according to our component model. Each UI capability should carry *view bindings* in SMCDL. Based on the capabilities of components, capabilities of arbitrary composition fragments result from the capabilities and communication channels of components. Their connections defined both within components and by IWC can be analyzed to derive a capability graph as presented in [1]. Such a graph consists of capability links as edges and capabilities as nodes. Capability links represent inter-component communication channels and intra-component relations as defined in SMCDL.

In light of these modeling concepts, several constellations of capabilities and view bindings must be considered by explanation techniques based on capability graphs. Figure 7 summarizes main cases which we refer to throughout the concept description.

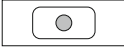
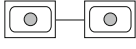
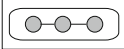

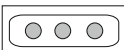


Case ID	Description	Schematic representation	Case ID	Description	Schematic representation
Single UI Capability			Capability Chains		
1	One view binding with one atomic operation		5	Single capability link between two Capabilities. Cases 1-4 apply as well.	
2	One view binding with one sequential operation consisting of several atomic operations, which have to be performed in a specific order.		6	Multiple ingoing capability links (OR Pattern): Data for a target capability are provided by several capabilities. Cases 1-4 apply as well.	
3	One view binding with one parallel operation expressing that several atomic operations have to be performed without specific order.		7	Multiple ingoing capability links (AND Pattern): Several capabilities are required in conjunction for a target capability. Cases 1-4 apply as well.	
4	Several view bindings representing alternatives. Cases 1-3 apply as well.				

Fig. 7. Main constellations to be considered (adapted from [4]).

Isolated subgraphs that are coherent in themselves are called capability chains. From a capability graph, an overlaying hierarchy graph is calculated per capability chain [1]. In this process, composite capabilities representing higher level functionalities offered by a CWA are estimated. To identify the interaction steps for a composite capability, it is necessary to recurse the hierarchy graph until reaching atomic capabilities. This leads to the corresponding capability chains, allowing to apply the cases introduced above.

5.2 Core Visualization Concepts

The proposed explanation techniques directly operate within the UI of components rather than in abstract views. To achieve this in context of our black-box component model, view bindings declaratively define links between UI and capabilities. Three basic visual element types are consistently used within the explanation techniques in order to cover all cases outlined in Fig. 7.

View binding Frames. Highlight UI elements referenced by view bindings.

Arrows. Connect view binding frames and represent capability links whereby they indicate data flow direction.

Description Boxes. Display textual explanations of capabilities and interaction steps. They are positioned near view binding frames or arrows, minimizing overlap. The text is context-sensitively generated from capability and view binding annotations.

Per default, a view binding frame is created for each DOM element referenced by the selectors of atomic view binding operations of a capability c . If c is a system capability or offers no view binding, the entire component panel is framed. In the special case of non-UI components a description box is rendered and framed. UI elements that are referenced by view bindings but invisible at the time needed, e. g., when a component uses tabs, pose further challenges. It may technically be possible to make such elements visible, e. g., via declarative instructions in SMCDL or imperative implementations that are invoked by the MRE. However, this leads to high effort for component developers and it seems questionable whether such solutions work generically. Thus, we employ a simplified, generic solution: Should a referenced element be invisible, the whole component panel is framed and a textual hint is shown in description boxes.

5.3 Assistance Features

We propose a set of generic explanation techniques that are based on the capabilities described in SMCDL, the capability and hierarchy graphs of a CWA, and the visual element types introduced above. These are: *Capability panels* enumerating capabilities of mashups and components; a *recommendation panel* that displays recommended patterns emphasizing their capabilities; the *inspection mode* for exploration of capabilities and capability links in the UI of components; and the *tutorial mode* which presents step-by-step instructions; as well as the *awareness mode* indicating data flow at runtime.

Inspection Mode. The *inspection mode* allows non-programmers to explore the functionality of components and CWA. Its main purpose is to present capability chains of a mashup, describe their functionality and indicate corresponding UI elements. This way, the inspection mode can be understood as CapView integrated within the live view.

The inspection mode can be activated and deactivated via a dedicated menu button. Once it is active, all inter-component capability links are visualized

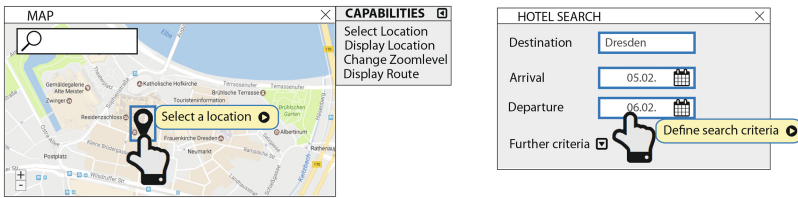


Fig. 8. Exemplified inspection mode for single components.

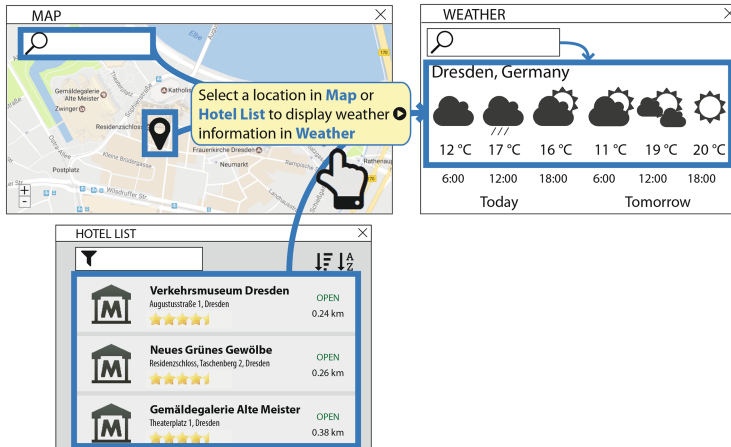


Fig. 9. Exemplified inspection mode for an inter-component capability [4].

by drawing view binding frames and arrows. Intra-component links are hidden initially in case they are not transitively connected to inter-component links, and are shown on demand. The rationale for this is to avoid confusing appearance. An additional reason is that component capabilities are often known to users.

When the user explores a component's UI, the corresponding capability panel appears. Furthermore, view binding frames become visible as soon as the associated UI elements are hovered as illustrated in the left part of Fig. 8. In cases 2 and 3, all other elements referenced in sister atomic operations are highlighted as well, cf. right side of Fig. 8. In addition, a description box appears and provides access to the tutorial mode for that capability, via a "play button". It contains a generated text that is derived from the capability re-using the concepts from [9]. If the capability whose view binding is hovered is part of a capability chain, all corresponding arrows and frames are emphasized, and intra-component chains become visible. Additionally, the description text mentions all capabilities of a link, pointing out cause and effect. To this end, sentences are formed from the capabilities and communication channels involved, paying attention to link direction and the hovered capability. We extended the label generator [9] to

also include component names and to connect several capability labels by “and” (case 7) and “or” (case 6). When hovering special text fragments, like component names and capability labels, component panel or view binding frame are highlighted.

As Fig. 9 shows, when selecting an arrow, it and all other arrows and view binding frames belonging to the represented capability link are emphasized and a description box is displayed. Again, users can watch a tutorial to learn in detail what to do.

We briefly mentioned in Sect. 3 that components can be arranged in different application views, requiring special considerations. In the inspection mode, components partaking in a capability chain but currently invisible are represented by their icon, which is positioned at the border and serves as view binding frame. Further, its description box allows to switch to the corresponding application screen.

Tutorial Mode. The *tutorial mode* focuses on explaining necessary interaction steps rather than the full detail of capabilities. Hence, UI capabilities are of main interest and are required to generate stepwise instructions. Tutorials can be accessed via buttons in capability panels and in description boxes of the inspection mode.

When the tutorial mode is activated for a capability or a capability chain, it visualizes and highlights the view binding frames of the first interaction step by graying out the remaining CWA, see Fig. 10. This way, user attention is directed. A description box appears, comprising a generated text as well as a navigation

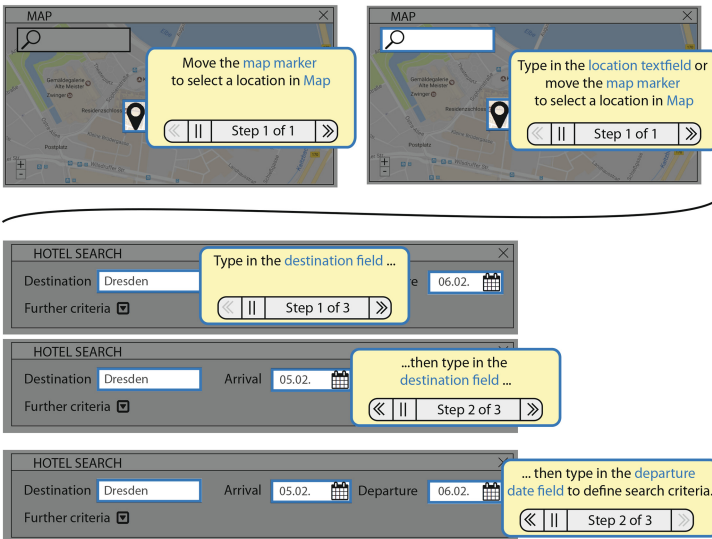


Fig. 10. Exemplified tutorial mode for a single component [4].

bar. The latter features controls to auto-play, close, pause, step forward, step back in the tutorial. The generated description texts mention the interaction technique and the element name as defined in the view binding, the capability achieved as well as the component name. Again, the sentence structure depends on the position of the capability in a capability link in order to properly reflect cause and effect.

Figure 10 illustrates example tutorials for cases 1 and 4 in its upper part. In case 2 one tutorial step per atomic operation exists, see lower part of Fig. 10, while in case 3 there is a single step and a text mentioning all atomic operations in conjunction. Similarly, in case 6 all source capabilities of a capability link are mentioned in one step, with one description box per source capability. In contrast, a constellation like in case 7 results in separate steps per source capability.

Tutorials also involve non-UI components. To do so, an additional step is included, that visualizes a description box displaying the component's name, icon and capabilities provided in context of the capability chain to be explained. For an example, please refer to Fig. 11. Analogously, components are handled that provide a capability to be explained in a step but that are not displayed in the current application view. In this case, the description box also allows to switch to the corresponding view and back.



Fig. 11. Exemplified tutorial mode for an inter-component capability [4].

Awareness Mode. The *awareness mode*'s purpose is to make users aware of data exchange through IWC when it occurs at run time of an arbitrary CWA. To this end, it builds upon the presented visualization concepts for inter-component capability links. By means of a dedicated button in the menu bar, the awareness mode can be activated.

Whenever an event is fired on a communication channel, the corresponding capability c is determined from the component description and saved as start capability c_s . In case c is no UI capability and linked with preceding capabilities, c_s is set to the first UI capability from the predecessors if there is one. Analogously c_t is obtained at the target side of a capability link. Then, an animation of configurable duration starts. It first displays the view binding frames of c_s as exemplified in the upper part of Fig. 12. Further, a description box with a generated text indicates the capability just achieved. Next, an arrow is shown, followed by the view binding frames of c_t and a corresponding description box, cf. lower part of Fig. 12.

Case 7, i. e., when several interactions are necessary in order to join all data before c_t can actually be performed, requires particular consideration. The last animation step is postponed and instead the view binding frames of the required other capabilities are shown and highlighted for a while, each complemented with a description box expressing necessary interactions. As soon as all input is provided, the animation continues with the frames and description boxes of c_t . In case a component providing c_s or c_t is not part of the current application view, we apply the approach known from the inspection mode: An icon represents the component and acts as view binding frame.

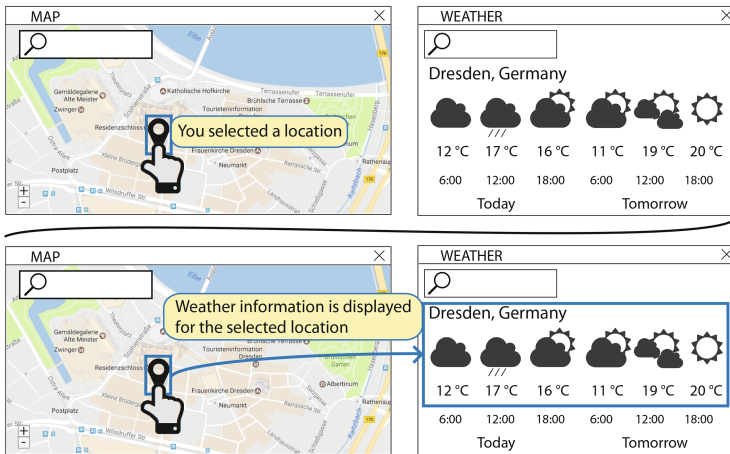


Fig. 12. Exemplified awareness mode for two instants [4].

Capability Panels. *Capability panels* give overview and insight on the capabilities of components or mashups. A capability panel represents the tree-like hierarchy graphs of a CWA or lists the capabilities of a single component, as indicated in Fig. 8. Therein, capabilities are textually described using generated labels. When a capability is selected by the user, the tutorial mode can be activated for it. Capability Panels are also synchronized with the inspection mode: When a user hovers a view binding there, affected capability panels highlight the corresponding capability.

Understanding Recommendations. A *recommendation panel* depicts recommended composition steps to users. As exemplified in Fig. 13, a novel two-step approach for visualizing recommendations takes place. Thereby, we largely build up on the capabilities of the underlying patterns. As a prerequisite, recommended patterns are clustered according to their capabilities. In a first step, all clusters are presented to the user with the help of generated labels, e.g., **display weather info**. When the user made his choice, descriptions for each pattern from that cluster appear. These contain component icons and additional text fragments, that supplement the cluster label to form a short sentence. The description mentions component names and the corresponding capabilities. When hovering these text parts, the component is highlighted or the view binding frame appears, respectively. In the previous example, a pattern-specific text extension may be ‘in Weather Forecast by the location selected in Map’.

To sum up, we introduce novel, generic explanation techniques that help non-programmers to explore, to learn how to use and to become aware of CWA capabilities.

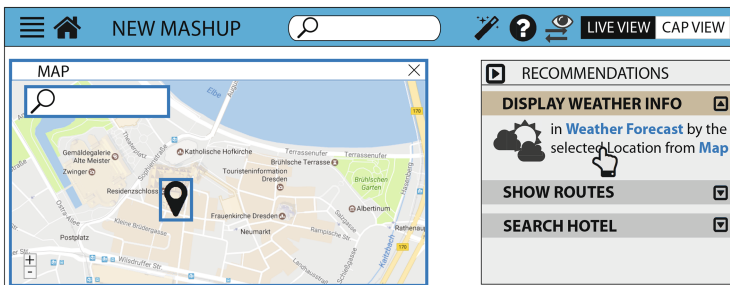


Fig. 13. Example recommendation menu (left side) with extension suggestions for a CWA [4].

6 Evaluation

Details on the prototype we developed within the CRUISE mashup platform and the user study we conducted for validation and evaluation are subject to this section.

6.1 Implementation

We implemented a prototype of the proposed explanation techniques as part of the CRUISE platform. Specifically, we extended the client-side of our MRE, which is developed using web technologies like HTML5 and CSS3. Several JavaScript frameworks are employed. For example, Bootstrap Popovers² serves for rendering description boxes and jsPlumb³ is used for drawing arrows based on SVG. Bootstrap Tour⁴ offers the necessary functionality to implement stepwise tutorials. The capability and hierarchy graphs of the current CWA are calculated by a dedicated web service and managed by the MRE as a JSON object. It is analyzed in order to setup corresponding data structures for the explanation techniques. Thereby, capability chains are processed in several ways: Intermediate system capabilities, e. g., in case of non-UI components, are skipped and information about composite capabilities, multiple view bindings of a capability and parallel or sequential operations are used to derive all interaction steps. Building up on this, Bootstrap Tours are instantiated and configured accordingly. Upon activation, the prototype of all three modes creates a transparent overlay `div` element per component. It serves as canvas for rendering view binding frames, description boxes and arrows. Further, we implemented the CWA capability and recommendation panels. Figure 14 illustrates the tutorial mode, explaining the capability `display weather` in case of a rather simple mashup of two components.

6.2 User Study

Methodology. We conducted a user study in order to evaluate the proposed concepts presented in Sect. 5. Ten non-programmers from very different background and sectors, e. g., forestry, mechanical engineering, photography, economics, sale and medicine, participated. They were in average 31 years old, ranging from 16 to 56. See Tables 1 and 2 for details. We created two groups, each with five persons. Group A used our mashup platform without, group B with the explanation techniques. According the think-aloud method, all participants were asked to individually solve several tasks and thereby articulate their thoughts about next required steps, expected system behavior and how the actual system response matches their expectations. Multiple tasks of increasing complexity in context of three different CWA had to be solved by our participants.

² <http://getbootstrap.com/javascript/>.

³ <https://jsplumbtoolkit.com/>.

⁴ <http://bootstraptour.com/>.

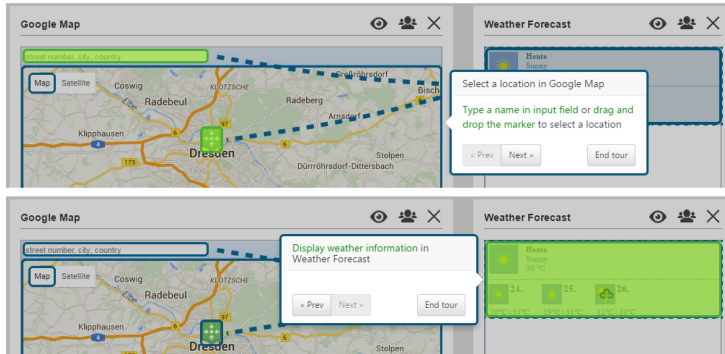


Fig. 14. Prototypical tutorial mode within the CRUISE mashup runtime environment.

1. The first application served for showing a weather forecast and therefore comprised a weather widget and a map.
2. Furthermore, a mashup composed of two maps and a route search widget allows to plan routes.
3. The most complex application, shown in Fig. 15 is built of 8 components that offer three overall mashup capabilities: searching routes, displaying weather and searching points of interest.

Simpler tasks demanded from participants to identify and achieve component capabilities, e. g., selecting a location in map, which can be done in several ways. Further, we tested if the users were able to identify and perform capabilities provided by component interplay. For example, participants were asked to find routes using the second CWA, and to search for points of interest with the help of the third mashup. As a further example, they had to identify the overall capabilities of the third CWA and the components that partake. Next, we tested

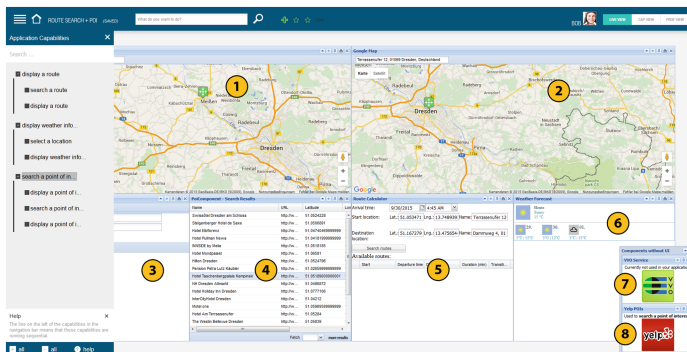


Fig. 15. Screenshot of a CWA utilized in our user study.

if participants are aware of IWC during mashup usage, for instance, between POI list ④ and map ① in the CWA shown in Fig. 15.

We not only created the think-aloud protocols, but additionally measured the number of successfully solved tasks per participant. To get a widely accepted and comparable usability measure, we further asked each person to fill out a SUS questionnaire. Finally, users were encouraged to freely comment on things they liked or disliked.

The main goal of our study was to show that the explanation techniques help participants to gain insight about given CWA. This should lead to increased task performance of group B, indicated by the number of successfully solved tasks. In order to analyze if our claim is true, we compared the results of both groups then.

Results and Discussion. The resulting SUS scores as well as the percentage of successfully solved tasks T_s are illustrated in Table 1 for group A and Table 2 for group B. As can be seen, T_s differs between both groups. Without the help of explanation techniques, participants of group A were in average able to solve 50% of the tasks, ranging from 36% to 59%. Members of group B were more successful, as they reached an average of 79% with a minimum of 64% and maximum of 100%. From the SUS questionnaires we learned, that participants of group A attested our prototype a lower usability than those of group B: We calculated an average SUS score of 57 for group A and 74 for group B.

Table 1. Results for group A (plain mashup platform).

Participant	Sector	Gender	Age	T_s	SUS
P1	Engineering (student)	Male	25	55%	60
P2	Warehouse Management	Male	56	36%	48
P3	Medicine	Female	45	55%	58
P4	Photography	Female	23	55%	65
P5	Mechanical Engineering (student)	Male	22	59%	63
Average			34	50%	57

We interpret these results as an indication, that our concepts seem useful and actually help non-programmers to better understand CWA they are unfamiliar with. An average SUS score of 74 attests good usability. We consider this as a satisfactory outcome taking the preliminary status of our prototype into account. From the think-aloud protocols and comments we identified potential improvements especially of our prototype. For example, members of group B repeatedly tried to directly perform steps when these were presented in a tutorial. However, this is impossible due to the invisible `div` element. Further, minor usability problems like inadequate icons became obvious. Not surprisingly, participants did not pay much attention to the capability panels of components they are

Table 2. Results for group B (using our concept).

Participant	Sector	Gender	Age	T_s	SUS
P6	Mechanics	Male	23	73%	73
P7	Sale	Female	52	64%	73
P8	Photography	Female	25	100%	78
P9	Forestry (student)	Female	23	73%	75
P10	Economics (student)	Male	16	86%	70
Average			28	79%	74

familiar with. In addition, some users revealed the limited support for parallel sequences in our prototype. Such shortcomings are subject to future work.

In summary, the results of our user study are promising and show that the presented concepts can improve users' understanding of CWA functionality and awareness of inter-widget communication. The participants which were assisted by explanation techniques were more successful in identifying capabilities, functional dependencies of components and necessary interaction steps. However, there are threads to validity. For example, the rather small number of participants seems suitable to identify usability problems [20], but larger-scale studies are needed to confirm our results. Further, we did not investigate if there are side-effects from other platform features that may especially have hindered the participants of group A. Our proposed concepts show limitations, like the simplistic solution for hidden elements. More importantly, component annotations have to be provided, which may be a demanding task, also due to different styles (amount and granularity of capabilities and view bindings) allowed, leading to a multitude of options to describe circumstances. However, in our opinion the presented concepts offer a lot of benefits. They function for black-box components and arbitrary CWA. Further, the capabilities of mashups can now be explored within the component UI in a systematic way rather than in a trial and error style. Tutorials can be generated for CWA, lowering the effort for component developers and resulting in a consistent appearance for all components. Tutorials offer contextualized instructions and operate directly in components' UI. As indicated by our evaluation, the explanation techniques help users to be aware of and to understand the capabilities of mashups.

7 Conclusions

Composite web applications and end user development complement each other quiet well. Developing and using such applications is still challenging for non-programmers, though. Users often lack awareness of IWC and understanding of the composite nature of CWA and their functionality. Recent mashup platforms offer no or very limited assistance and proper tooling in this regard, which results in trial and error strategies of non-programmers. Our mashup approach features

interwoven runtime and development time and a palette of appropriate EUD tools continuously supporting non-programmers. This way, we aim to overcome limitations of existing mashup platforms.

In this paper, we introduce our novel explanation techniques and give insights on the iterative design process. It incorporates scenarios and lessons learned from earlier concepts and the associated prototypes as well as conducted user studies. As a result, we propose a generic approach fostering exploration, explanation and awareness of the functionality of arbitrary CWA. Our conceptual foundation are components annotated with provided capabilities and their view bindings. Utilizing this, we visualize what single components are capable of and which interactions users have to perform directly within the component UI. We apply these core concepts to capabilities of composite web application, that result from component interplay through inter-widget communication. An overview panel lists mashup capabilities and step-by-step tutorials are generated. They interactively and animatedly present required interaction steps to non-programmers and complement highlighting UI elements with textual descriptions. Furthermore, these visualization and interaction techniques are re-used to foster users' awareness of IWC when it occurs at run time. Finally, our concepts allow non-programmers to comprehend the effects of recommendations. We pointed out some limitations of our concepts. However, as indicated by our evaluation results, the proposed solutions can actually help non-programmers to better understand CWA.

In future work we plan to perform the next cycle in our user-centered design process, i. e., to adapt our prototype according to feedback and lessons learned as well as to evaluate the next iteration on a larger scale. Further, we aim create a live view extension enabling composition tasks with the help of core concepts presented in this paper.

Acknowledgements. The work of Carsten Radeck is funded by the European Union and the Free State of Saxony within the EFRE program.

References

1. Radeck, C., Blichmann, G., Meißner, K.: Estimating the functionality of mashup applications for assisted, capability-centered end user development. In: Proceedings of the 12th International Conference on Web Information Systems and Technologies (WEBIST 2016), pp. 109–120. SciTePress (2016)
2. 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). https://doi.org/10.1007/978-3-642-39200-9_11
3. Tschudnowsky, A., Pietschmann, S., Niederhausen, M., Hertel, M., Gaedke, M.: From choreographed to hybrid user interface mashups: a generic transformation approach. In: Casteleyn, S., Rossi, G., Winckler, M. (eds.) ICWE 2014. LNCS, vol. 8541, pp. 145–162. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08245-5_9

4. Radeck, C., Meißner, K.: Helping non-programmers to understand the functionality of composite web applications. In: Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST 2017), pp. 109–120. SciTePress (2017)
5. Gery, G.: Attributes and behaviors of performance-centered systems. *Perform. Improv. Q.* **8**, 47–93 (1995)
6. Cao, C.: Helping end-user programmers help themselves - the idea garden approach. Ph.D. thesis, Oregon State University (2013)
7. Ko, A.J., Myers, B.A.: Designing the whyline: a debugging interface for asking questions about program behavior. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2004, pp. 151–158. ACM (2004)
8. Kuttal, S.K., Sarma, A., Rothermel, G.: Debugging support for end user mashup programming. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2013, pp. 1609–1618. ACM (2013)
9. 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). https://doi.org/10.1007/978-3-642-39200-9_14
10. Picozzi, M.: End-user development of mashups: models, composition paradigms and tools. Ph.D. thesis, Politecnico di Milano (2013)
11. Aghaee, S., Pautasso, C.: End-user development of mashups with NaturalMash. *J. Vis. Lang. Comput.* **25**, 414–432 (2014)
12. Lizcano, D., Lóez, G., Soriano, J., Lloret, J.: Implementation of end-user development success factors in mashup development environments. *Comput. Stand. Interfaces* **47**, 1–18 (2016)
13. Mehandjiev, N., Namoun, A., Lécué, F., Wajid, U., Kleanthous, G.: End Users Developing Mashups. In: Bouguettaya, A., Sheng, Q., Daniel, F. (eds.) *Web Services Foundations*, pp. 709–736. Springer, New York (2014). https://doi.org/10.1007/978-1-4614-7518-7_28
14. Liu, X., Ma, Y., Huang, G., Zhao, J., Mei, H., Liu, Y.: Data-driven composition for service-oriented situational web applications. *IEEE Trans. Serv. Comput.* **8**, 2–16 (2015)
15. Chen, H., Lu, B., Ni, Y., Xie, G., Zhou, C., Mi, J., Wu, Z.: Mashup by surfing a web of data apis. *Proc. VLDB Endow.* **2**, 1602–1605 (2009)
16. Trinh, T.D., Wetz, P., Do, B.L., Kiesling, E., Tjoa, A.M.: Semantic mashup composition from natural language expressions: preliminary results. In: Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services (iiWAS 2015), pp. 44:1–44:9. ACM (2015)
17. Ghiani, G., Paternò, F., Spano, L.D., Pintori, G.: An environment for end-user development of web mashups. *Int. J. Hum Comput Stud.* **87**, 38–64 (2016)
18. Radeck, C., Blichmann, G., Mroß, O., Meißner, K.: Semantic mediation techniques for composite web applications. In: Casteleyn, S., Rossi, G., Winckler, M. (eds.) ICWE 2014. LNCS, vol. 8541, pp. 450–459. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08245-5_30
19. Radeck, C., Lorz, A., Blichmann, G., Meißner, K.: Hybrid recommendation of composition knowledge for end user development of mashups. In: Proceeding of the ICIW 2012, the Seventh International Conference on Internet and Web Applications and Services, pp. 30–33. XPS (2012)
20. Nielsen, J.: Why you only need to test with 5 users (2000). Accessed 02 Jan 2017

Author Index

- Alsubai, Shtwai 1
- Bernardino, Jorge 45
- Biørn-Hansen, Andreas 64
- da Silva, Clay Palmeira 115
- De Meuter, Wolfgang 132
- Demurjian, Steven A. 221
- Devoegele, Thomas 115
- Faes, Axel 87
- Gnirke, Lukas 221
- Grønli, Tor-Morten 64
- Kambona, Kennedy 132
- Lamotte, Wim 87
- Latif, Atif 161
- Limani, Fidan 161
- Majchrzak, Tim A. 64, 202
- Marx, Robin 87
- Meißner, Klaus 249
- Messai, Nizar 115
- Mikkonen, Tommi 21
- North, Siobhán 1
- Paz, Solange 45
- Pelucchi, Mauro 179
- Psaila, Giuseppe 179
- Quax, Peter 87
- Radeck, Carsten 249
- Renaux, Thierry 132
- Rieger, Christoph 202
- Rivera Sánchez, Yaira K. 221
- Sam, Yacine 115
- Taivalsaari, Antero 21
- Toccu, Maurizio 179
- Tochtermann, Klaus 161
- Wijnants, Maarten 87