Awareness and Control for Inter-Widget Communication: Challenges and Solutions

Olexiy Chudnovskyy[1], Stefan Pietschmann[2], Matthias Niederhausen[3], Vadim Chepegin[5], David Griffiths[4], and Martin Gaedke[1]

[1] Chemnitz University of Technology, Germany
{olexiy.chudnovskyy,gaedke}@informatik.tu-chemnitz.de
[2] Technische Universität Dresden, Germany
stefan.pietschmann@tu-dresden.de
[3] T-Systems Multimedia Solutions GmbH, Germany
matthias.niederhausen@t-systems-mms.com
[4] University of Bolton, UK
d.e.griffiths@bolton.ac.uk
[5] TIE Nederland B.V., the Netherlands
vadim.chepegin@tiekinetix.com

Abstract. Recently, widget-based Web applications, i. e., mashups have gained momentum, as they make it possible to address the "long tail" of software needs. By enabling data and control flow among widgets – inter-widget communication (IWC) – integration of data and functionality can be defined by the end users themselves. However, IWC entails several problems that may reduce the overall user confidence in a system. Based on the results of user studies on the OMELETTE mashup platform, this paper analyzes the problem space and evaluates possible solutions to improve user perception of IWC. Further, a discussion of promising techniques is offered and pending challenges are identified.

Keywords: mashup, widget, inter-widget communication.

1 Introduction

The vision of users who drag-and-drop and combine applications from any location on the Web, local drive, or cloud storage, in their own workspaces has never been so close to becoming a reality. The modern Web offers powerful mashup platforms which enable end users to create their own situational applications on the fly without the intervention of developers. Research on such *User Interface Mashups* (UI Mashups) has made significant progress towards this vision. One of the most important concepts in this field is that of *widgets* – interactive components which provide the end user with access to data, services, and application logic. A number of initiatives have been proposed, addressing the emerging need for simple, flexible, and powerful composition environments, e.g. [8,12].

Their main goal is to enable end users to aggregate data and functionality from various sources on one screen or *workspace*. One of the key features of these platforms, *inter-widget communication* (IWC), allows widgets to exchange events and data. Depending on the communication paradigm, solutions differ in their degree of *automation* (manual effort may be required to establish connections), *end user suitability* (usability and complexity of IWC configuration vary) and *generality* of the approach (e. g., compatibility between widgets of different vendors is not guaranteed).

The success of IWC-aware platforms is highly dependent on the degree to which they support the above characteristics. Although all of them are significant and desirable from the end users' point of view, in practice it is hard to optimize all three simultaneously. Moreover, recent research on domain specific mashups shows that, to some extent, *generality* contradicts *end user suitability* [4].

The OMELETTE project [2] has been working on finding an appropriate trade-off between the first two aspects, namely automation and end user suitability. The results of this work were incorporated into a mashup environment with IWC implemented by means of a publish-subscribe messaging pattern (pub/sub). A recent user study [13] conducted with 44 participants in Germany and China revealed a number of issues, some of which are the result of the underlying mashup approach. The goal of this paper is to elaborate on these findings by presenting a survey of approaches in which similar problems have been tackled and discussing the most promising techniques in the context of mashup platforms.

2 Towards End-User Friendly IWC: Existing Challenges

In contrast to other approaches, in which mashup developers have to deal with abstract control flow and data flow models, in OMELETTE there is no difference from a user perspective between design and execution. Mashup composition takes place at run-time and its results are immediately evident to users. A distinct feature of OMELETTE is that users are not required to establish explicit connections between widgets. Communication, i. e., data flow emerges as soon as widgets are placed together within a workspace. This is achieved by means of the messaging bus: widgets subscribe and publish messages on different communication channels, known as *topics*. The decision to apply pub/sub was motivated by recent findings, which highlighted the importance of working "out-of-the-box" [5] and the usability issues of wiring approaches for end users [10].

Thus the OMELETTE solution implies that the complex issues of composition and wiring widgets are best left to skilled developers of widgets. This, however, implies a lack of *awareness* and *control* by end users – an implication which was confirmed in the interviews and observations during the user studies. Thus, the focus of this paper is on challenges and problems from an *end user perspective*.

2.1 Problem Space: Awareness

The first problem space comprises the challenges that users face when interacting with a pre-defined workspace. It may be split into the following sub-categories:

Cold Start Problem. Upon opening a workspace, end users do not know which of the widgets are actually inter-connected. Users have to learn the data and control flows as they use a mashup and explore it. While in general this may merely frustrate users, such "exploratory" interaction can also affect live data, causing undesired side effects.

IWC Transitivity. Similarly, it is very hard for users to distinguish direct and transitive connections between widgets. The latter occur when one widget triggers action in another, which in turn triggers a third widget. While this behavior may be intended, it can negatively affect users in their understanding of the resulting functionality: First, users may see relationships when there are none, and misinterpret their findings. Second, widget reuse and workspace refactoring will result in unpredictable behavior, e. g., the loss of functionality whenever the "linking" widget is missing.

Data Ignorance. Users typically do not see the data being transferred between widgets. Instead, they only perceive the effects of their transfer, i. e., that a receiving widget is updated with new data. While one can argue that providing this information may overburden users, data ignorance still leads to three problems: users can only guess which widgets are compatible and work together; mistaken expectations of the data transferred lead to misinterpretations of the application behavior; possibly untrusted widgets might receive sensitive data without the user's consent.

2.2 Problem Space: Control

The second set of problems stems from the need of users to modify how a mashup works. IWC connections established implicitly, i. e., based on the pub/sub paradigm, greatly simplify the start of work with a mashup but also lead to a loss of control.

Lack of Extensibility. Users cannot explicitly establish new connections between widgets. Due to the potentially large number of widgets developed by independent parties, it is both impossible to foresee all valid widget combinations and impractical to try and guarantee their interoperability. Thus, users will often want previously unconnected widgets to work together and to establish a link between them manually.

Rigidity. In pre-configured workspaces, it may be necessary to change the control or data flow, i. e., the way widgets are connected by default. This can be supported in its full complexity, as with wiring tools, or by offering more subtle actions, such as allowing users to isolate widgets as senders or receivers of data. There are many possible reasons for this, e. g., because a widget is untrusted, does not work as expected, or simply because it should hold an intermediary result to be saved for later.

Clunkiness. Establishing a temporary data flow can be desirable and more convenient than setting up a permanent connection. The studies revealed that many end users intuitively work with the data in the widgets by trying to drag-and-drop from source to receiver. This user-triggered temporary data flow is usually not foreseen by IWC mechanisms – be they wiring or pub/sub approaches – and platforms.

Addressing these challenges is crucial in order to boost end user acceptance and to promote the use of widget mashups in business environments. The next section will evaluate possible solutions.

3 Analysis of Existing Approaches

The following survey presents state-of-the-art techniques from the End User Development domain (EUD) in the context of the above problem spaces. The approaches are compared based on the degree of technical skill required by end users to employ them in mashups.

3.1 Solutions for Problem Space: Awareness

Self-Descriptive Design (SDD). Systems employing SDD mechanisms try to make users aware of functional dependencies between widgets at the application layer by the means of annotations or visual markup. Whenever users are confronted with new (e.g., shared) mashups, looking at individual widgets very often does not provide the "big picture", i. e., the overall functionality. SDD-based approaches address this problem by making mashups as self-explanatory as possible. Therefore, they provide annotation tools to be used in the phase of mashup creation. In [3], the authors suggest to make internal knowledge explicit by usage of *implicit*, *explicit* and *literate* annotations.

Additive Views (AV). One of the common practices for increasing user awareness in software systems is to provide various views on the application. Using suitable metaphors, these views enable users to explore the internal characteristics of the application, i.e., structure, components, data and control flows. Additive views are usually implemented either in an *integrated* or in a *separated* fashion. Integrated views try to avoid the "break" between usage and programming modes. In the EDYRA mashup environment [12], a running mashup can be augmented by dedicated overlays. Users are also able to highlight direct and transitive connections between components, raising the overall awareness of IWC in the mashup.

Surprise-Explain-Reward Strategy (SER). A surprise-explain-reward strategy aims at communicating non-obvious behavior of a system to end users and letting them engage in further exploration activities [14]. Information about inscrutable activities appears in ways that grab users' attention (*surprise*) and entice them to learn more about the causes. An appropriate help system supports the learning process and opens new perspectives on the possibilitites of the system (*explain*). Having applied the newly learned technique, users benefit from advanced platform capabilities (*reward*). In the Forms/3 project [1], this idea has been applied to ensure data integrity in end-user-created spreadsheets.

Question Asking Strategy (QA). This strategy is applied to find the causes of unexpected or non-obvious application behavior. Based on explicit knowledge about the structure of an application, the system is able to provide answers to specific types of user inquires. A dialog often takes place in natural language and does not require the user to learn any programming formalisms or debugging techniques. The WhyLine tool [6] applies this technique to enable unskilled developers to test their algorithms. Using menus and pictograms of objects involved, users can construct "why did" and "why did not" questions in order to explore the system's behavior. A user study showed that developers were more efficient with this system than with traditional debugging tools. The HANDS project [9] conducted several user studies in order to understand how people without programming skills think of and express software design. After implementing their findings using natural language in question building and answering, the authors claimed that even ten-year-olds were able to create meaningful programs.

3.2 Solutions for Problem Space: Control

Parametrization (PAR). Along with interface customization, *parametrization* is one of the simplest and most common forms of EUD. It assumes that software is designed in a way that enables modification of its behavior by changing the values of a pre-defined set of parameters, e.g. the location of a news feed. Netvibes and iGoogle successfully employ the mechanism in widget-based dashboards. The way in which the parametrization view is exposed differs: it can be offered by widget developers or by the composition platform. The latter is done by portals based on explicit parameter declarations in widget descriptors. Netvibes and iGoogle support both parametrization modes.

Programming by Demonstration (PBD). This is a well-proven technique that enables end users to specify desired functionality by providing examples of its behavior [7]. Based on demonstrated activities and data samples, a PBD system tries to generalize user actions and to derive an algorithm. One of the open challenges facing PBD systems is how to represent the captured algorithm and to facilitate its future adaptation by end users. The CRUISe project [11] proposes an extension to the interface between widgets and mashup platform. Widget authors can notify the platform of user interactions, e. g., when users drag data beyond a widget's perimeters. The platform monitors further user interactions, e. g., the data being dropped onto other widgets. This way, users can implement ad hoc data exchange and also establish permanent connections.

Programming by Specification (PBS). This comprises EUD approaches that enable users to create mashups by defining the data/control flow themselves. This process of EUD is predominantly based on *visual programming* languages involving metaphors such as "Lego" constructors or electrical circuits. Similar techniques are used in the majority of mashup platforms, such as Yahoo Pipes[1] or JackBe Presto[2]. In [4], the authors propose sacrificing the generality of mashup tools in favor of simplicity and comprehensiveness of the system by applying

[1] `http://pipes.yahoo.com`
[2] `http://mdc.jackbe.com/products/mashboard.php`

domain-specific composition tools. A user study of the ResEval platform has confirmed this assumption, showing that end users understand the composition paradigm and can master the development of mashups if they are focused on single domains and unburdened from data transformation issues.

4 Comparison of Approaches and Drawn Guidelines

Table 1 presents a comparison of the previously discussed techniques based on the expert evaluation.

Table 1. Applicability of EUD techniques to the widget mashup domain

Criteria/Approach	SDD	AV	SER	QA	PAR	PBD	PBS
Cold Start Problem	○○	●	–	●●	–	–	–
IWC Transitivity	○○	●	●●	●●	–	–	–
Data Ignorance	○○	○○	●●	●●	–	–	–
Rigidity	–	–	–	–	●●	○	●
Clunkiness	–	–	–	–	–	●●	●
Lack of Extensibility	–	–	–	–	–	●●	●

●● – applicable without deep understanding of data types and control flows, ● – applicable with basic knowledge on data types and control flows, ○○ – limited applicability without deep understanding of data types and control flows, ○ – limited applicability with basic knowledge on data types and control flows, — – not applicable

The *Surprise-Explain-Reward* strategy differs from the Question-Asking approach in that users are notified about internal mashup activities right before or right after they happened. This implies that the *cold start problem* is not addressed appropriately, i. e., users are unable to explore connections or exchanged data before the real communication takes place. This disadvantage can be crucial for cost- or load-causing widgets. The approach requires the platform to include appropriate notification mechanisms and an explicit declaration of the mashup structure including widget capabilities and IWC configuration.

Additive Views can address all of the awareness-related problems by enabling end users to explore the internals of a mashup at any time. The main challenge here is to find a compromise between complexity and usability, i. e., to identify suitable abstractions and to adjust the view according to user skills. Recent research proposes to implement overlay views to lower the cognitive load while working with alternative mashup representations [12]. Some familiarity with the "wiring" concept is required to understand connections. To facilitate AV, the platform needs to access mashup configuration and widget interface descriptions.

The applicability of *Self-descriptive Design* is constrained as it is hard to design descriptive graphics for a mashup if the screen size is not fixed. Accordingly, the layout of mashups is not completely consistent between platforms, and it is not possible to predict the degree to which users can change the position of widgets. The adoption of this approach implies (a) that the container makes design tools available to the author, (b) that the design of the container needs to be considered at the same time as the design of mashup functionality, and (c) that this work will need to be repeated whenever the mashup is deployed in a new container.

Within the *Control* problem space, *Parametrization* is the most promising approach with a focus on end users. Although it does not cover the *lack of extensibility* and *clunkiness* of a mashup, the *rigidity* of composition can be influenced if the IWC capabilities of a component are configurable. This, however, goes at the expense of simplicity for the user. To lower the learning curve, all configuration options should be exposed in a uniform manner, e.g., by avoiding all widget-internal configuration dialogs.

Programming by Demonstration addresses the *lack of extensibility* and *clunkiness* problems and enables the definition of new communication paths in an end-user-friendly way. Drag-and-drop has been successfully applied in many instances and is well understood by end users. Also, observation of user interaction with a mashup can be utilized to derive new connections between widgets. In the context of the *rigidity* problem, PBD poses new challenges, such as end-user-friendly representations of generalized algorithms and appropriate modification facilities. Additionally, user interactions with widgets have to be made explicit, e. g., by notifying the observation engine about starting drag'n'drop operations.

Programming by Specification strives to enable end users to design and modify existing software artifacts. However, in targeting all three problems from the *Control* problem space, it assumes that users are able to write behavior specifications and are familiar with basic programming concepts. Projects which utilized this technique have achieved varying degrees of usability. Environments based on natural languages and domain specific vocabularies were more efficient and comprehensive for end users than general purpose composition tools. To apply PBS efficiently, supportive EUD techniques such as instant feedback, decision support and integrity checks should be incorporated into the system.

Based on the above analysis, the following suggestions are made regarding the combination of techniques to address the identified problems:

Cold Start Problem. Provide overlay views on the widget composition, visualizing possible communication paths (AV). These views can be layered (one layer per widget) to use the screen estate efficiently. A help system can also be provided, enabling the user to explore the composition through questions in natural language (QA).

IWC Transitivity. In the overlay view, enable users to discern the direction of communication paths. During data transfer between widgets, visualize active communication paths and enable their exploration and configuration (SER). Empower the help system to answer questions in natural language regarding directions of IWC paths (QA).

Data Ignorance. Enable users to explore possible data flows within the IWC overlay view (AV). During active communication, notify users about ongoing data exchange and enable exploration or modification of this communication path (SER). Extend the help system to answer questions regarding data being transferred between widgets (QA).

Lack of Extensibility. Use observation of user-widget interactions to derive new possible connections between widgets (PBD).

Rigidity. Provide enable/disable parametrization of communication paths and the possibility of isolating widgets from IWC (PAR).

Clunkiness. Provide a drag-and-drop infrastructure to enable one-time communication between widgets (PBD).

5 Conclusions

This paper demonstrates that end-user friendly IWC is needed, but is also difficult to achieve. To tackle this problem, the typical challenges for IWC solutions were derived from user studies conducted within the OMELETTE project and literature review. Based on the findings, the next steps will be to implement the chosen IWC mechanisms using as a basis the open source OMELETTE platform and to evaluate the new features with end users.

Acknowledgment. This work was supported by the European Commission (project OMELETTE, contract 257635).

References

1. Burnett, M., Atwood, J., Walpole Djang, R., Reichwein, J., Gottfried, H., Yang, S.: Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. Journal of Functional Programming 11(2), 155–206 (2001)
2. Chudnovskyy, O., Nestler, T., Gaedke, M., Daniel, F., Ignacio, J.: End-User- Oriented Telco Mashups: The OMELETTE Approach. In: WWW 2012 Companion, pp. 235–238 (2012)
3. Dinmore, M.: Documenting problem-solving knowledge: Proposed annotation design guidelines and their application to spreadsheet tools. In: Proceedings of EuSpRIG 2009, pp. 57–68 (2009)
4. Imran, M., Soi, S., Kling, F., Daniel, F., Casati, F., Marchese, M.: On the systematic development of domain-specific mashup tools for end users. In: Brambilla, M., Tokuda, T., Tolksdorf, R. (eds.) ICWE 2012. LNCS, vol. 7387, pp. 291–298. Springer, Heidelberg (2012)
5. Isaksson, E., Palmer, M.: Usability and inter-widget communication in PLEs. In: Proceedings of MUPPLE 2010 (2010)
6. Ko, A., Myers, B.: Designing the whyline: a debugging interface for asking questions about program behavior. In: Proceedings of CHI 2004, vol. 6, pp. 151–158 (2004)
7. Lieberman, H.: Your Wish is My Command: Programming By Example (Interactive Technologies). Morgan Kaufmann (2001)
8. Lizcano, D., Soriano, J., Reyes, M., Hierro, J.J.: A user-centric approach for developing and deploying service front-ends in the future internet of services. International Journal of Web and Grid Services 5, 155–191 (2009)
9. Myers, B., Pane, J., Ko, A.: Natural Programming Languages and Environments. Communications of the ACM 47(9), 47–52 (2004)

10. Namoun, A., Nestler, T., De Angeli, A.: Service Composition for Nonprogrammers: Prospects, Problems, and Design Recommendations. In: Proceedings of ECOWS 2010, pp. 123–130. IEEE (December 2010)
11. Pietschmann, S., Voigt, M., Meißner, K.: Rich communication patterns for mashups. In: Brambilla, M., Tokuda, T., Tolksdorf, R. (eds.) ICWE 2012. LNCS, vol. 7387, pp. 315–322. Springer, Heidelberg (2012)
12. Rümpel, A., Radeck, C., Blichmann, G., Lorz, A., Meißner, K.: Towards doityourself development of composite web applications. In: Proceedings of ITS 2011, pp. 330–332 (2011)
13. The OMELETTE Project (FP7/2010-2013 GA n 257635). D7.4 - evaluations of demonstrators report. Public deliverable (2013), http://goo.gl/o0JFG
14. Wilson, A., Burnett, M., Beckwith, L., Granatir, O., Casburn, L., Cook, C., Durham, M., Rothermel, G.: Harnessing curiosity to increase correctness in end-user programming. In: Proceedings of CHI 2003, pp. 305–312 (2003)