Inter-Widget Communication by Demonstration in User Interface Mashups

Olexiy Chudnovskyy[1], Christian Fischer[1], Martin Gaedke[1], and Stefan Pietschmann[2]

[1] Chemnitz University of Technology, Germany
{olexiy.chudnovskyy,christian.fischer,gaedke}@informatik.tu-chemnitz.de
[2] Technische Universität Dresden, Germany
stefan.pietschmann@tu-dresden.de

Abstract. User Interface Mashups have become increasingly popular, as they allow end users with little programming skills to create situational Web applications on their own. Those are built by composing interactive components, so-called widgets, whose integration is achieved by the means of "inter-widget communication" (IWC). Since widgets are built by different vendors and rely on different data models, IWC rarely works "out of the box", which leaves users with the tedious task of manual wiring and limited functionality.

This paper presents a semi-automatic, end-user friendly approach to extend widgets with IWC capabilities by employing the programming by demonstration paradigm. The solution is demonstrated using an extension of Apache Rave, an open-source widget composition platform.

Keywords: mashup, inter-widget communication, programming by demonstration.

1 Introduction

User Interface Mashups (UI Mashups) facilitate the aggregation of several widgets on a canvas or "workspace" to create situational applications. The integration of functionality and data offered by widgets is achieved by the so-called *Inter-Widget Communication* (IWC). The corresponding messaging infrastructure provided by many platforms allows for synchronization and message transfer between widgets which lets them act as one integrated solution with significantly improved user experience [3,4,2]. Many of the widgets currently available on the Web do not make use of IWC. Some of them are simply not designed to be used in compositions. Others rely on component models unaware of IWC mechanisms, such as W3C widgets. Finally, IWC-enabled widgets developed by different parties suffer from compatibility problems with regard to communication models and data formats. As result, users often have to input the same data multiple times and synchronize views manually.

In prior work [1] we have proposed a semi-automatic context-independent approach for extending widgets with IWC capabilities. It is targeted at domain

F. Daniel, P. Dolog, and Q. Li (Eds.): ICWE 2013, LNCS 7977, pp. 502–505, 2013.

experts and skilled users, as a basic understand of data types is required. The work presented here addresses users with little or no programming skills and provides the following major contributions: First, we show how Web-based widgets can be extended towards IWC capabilities on the graphical user interface (GUI) level automatically; Second, we demonstrate how IWC configuration can be done using the *programming by demonstration* (PBD) technique; Finally, we demonstrate how this approach has been integrated and tested with several open source projects.

The rest of the paper is structured as follows. After giving an overview of related works in the next section, Sect. 3 presents the proposed solution for end-user friendly IWC configuration. Finally, Sect. 4 concludes the paper and gives an outlook on future work.

2 Related Work

Building mashups by demonstration has been explored in the Karma project [6]. The project focused on so called data mashups, i.e. ones, which extract, integrate and display data from different sources. Users apply PBD technique to specify, how data from Web pages is extracted, normalized and combined together. In contrast, the focus of this work lies on widget-based mashups and thus requires further techniques to configure GUI-level IWC between widgets.

Geppeto project introduced the idea of programming on the GUI level and applied it the context of widget-based dashboards [5]. Using several special-purpose widgets and the PBD technique users are able to define workflows consisting of multiple GUI actions across different widgets. However, the recorded workflows can only be triggered by user or by pre-defined system events and not by widgets themselves.

Several research projects have focused on end-user friendly IWC configuration. Within the CRUISE project [4] users can establish connections between widgets by means of the drag&drop technique. However, widgets need to be designed this way and rely on semantically compatible data types. The solution presented in this paper is more generic, as widgets do not need to be IWC aware or to comply with any particular interface.

3 End-User Friendly IWC Configuration

The proposed concept is applied in the context of so-called choreographed UI mashups [7]. Therein, communication emerges without an explicit data flow definition: Widgets send and receive messages based on the publish-subscribe (pub/sub) messaging pattern. To be semantically compatible they utilize a reference ontology describing the data concepts shared. Widgets themselves are treated as black boxes with public interfaces exposing publications and subscriptions to certain topics and the data types involved.

Implementations of this model predominantly support application-level events and operations. GUI changes and interactions are usually not communicated via

pub/sub. Naturally, if widgets do not expose application-level interfaces or the concepts are incompatible (e.g., by not using a common reference ontology), IWC becomes impossible. Our approach addresses this problem by enriching the widget interface with events and operations at the presentation layer. These can be employed to orchestrate widget GUIs – guided by the user – thereby establishing connections between widgets.

To enrich widgets with the new interface we have extended the widget containers Apache Wookie[1] and Shindig[2] so that DOM-event listeners are automatically added to the source code of instantiated widgets, e. g., for HTML inputs, select boxes, buttons, and anchors. These extension mechanisms allow for an easy monitoring and invocation of state changes for the above-mentioned elements. To establish a "connection" between widgets, users perform GUI actions in one widget, which should *lead* to the message transfer, and actions in another widget, which should be executed *after* the message transfer. A learning system then detects correlations in recorded action sequences indicating possible data flow. One correlation currently supported in our prototype is the reoccurance of text in different HTML input elements. If a user searches for "London" in a weather forecast widget and right after that selects the same city in a map widget, the platform will detect this repeated input and derive a "connection" between the GUI elements. Fig. 1 illustrates this example workflow.

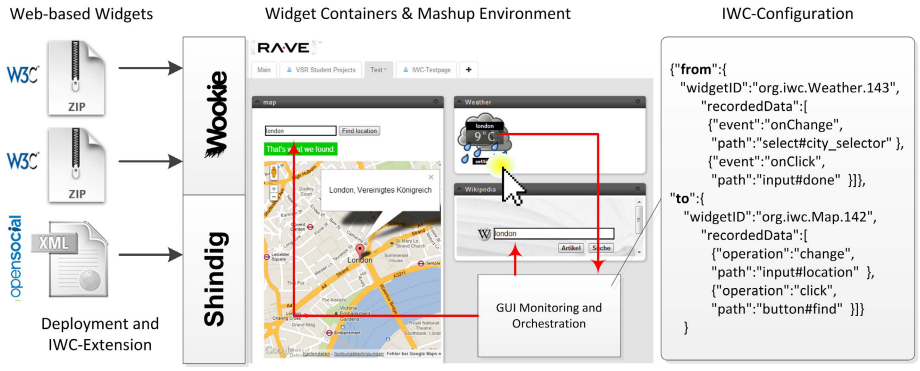

Fig. 1. Configuration of IWC using Programming by Demonstration

From then on, whenever a user starts a similar interaction with the source widget, the system will automatically complete the corresponding interaction in the target widget with the help of the automatically integrated code. The configuration is stored per workspace and user so that widgets can be reused in different contexts without prior source code modifications.

In the demo session we plan to showcase the above-mentioned trip planning scenario. First, we demonstrate how a map, a weather forecast, and a Wikipedia

[1] http://wookie.apache.org/
[2] http://shindig.apache.org

widget can be automatically extended towards IWC capabilities. Then, we show how the desired data flow can be configured by simply interacting with the aggregated widgets. Finally, we present the derived IWC configuration and automatic re-execution of the recorded actions.

A screencast of the planned demonstration and a running prototype based on Apache Rave[3] are available at `http://vsr.cs.tu-chemnitz.de/demo/iwc-pbd`.

4 Conclusions

This paper describes an approach to extend stand-alone widgets with IWC functionality in an end-user friendly fashion. To achieve this, widgets are automatically equipped with GUI-level observers, which allow for the deduction of logical connections by monitoring user interactions. As a result, widget integration does not require any programming skills. Users apply the same techniques as they do while naturally interacting with their Web applications.

The approach is currently limited to simple patterns of user interactions with a focus on Web-based forms. Future work will explore how to detect and transfer complex data between widgets. Further, we plan to conduct a user study to improve on the usability and scrutability of the approach.

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

References

1. Chudnovskyy, O., Müller, S., Gaedke, M.: Extending web standards-based widgets towards inter-widget communication. In: 4th Intl. Workshop on Lightweight Integration on the Web, pp. 93–96 (2012)
2. Chudnovskyy, O., Nestler, T., Gaedke, M., Daniel, F., Ignacio, J.: End-User-Oriented Telco Mashups: The OMELETTE Approach. In: WWW 2012 Companion Volume, pp. 235–238 (2012)
3. Lizcano, D., Soriano, J., Reyes, M., Hierro, J.J.: Ezweb/fast: Reporting on a successful mashup-based solution for developing and deploying composite applications in the upcoming ubiquitous soa. In: Proc. of the 2nd Intl. Conf. on Mobile Ubiquitous Computing Systems, Services and Technologies, pp. 488–495. IEEE (September 2008)
4. 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)
5. Skrobo, D.: Widget-Oriented Consumer Programming. AUTOMATIKA: Journal for Control, Measurement, Electronics, Computing and Communications 50(3-4), 252–264 (2009)
6. Tuchinda, R., Knoblock, C.A., Szekely, P.: Building Mashups by Demonstration. ACM Transactions on the Web 5(3), 1–45 (2011)
7. Wilson, S., Daniel, F., Jugel, U., Soi, S.: Orchestrated User Interface Mashups Using W3C Widgets. In: Harth, A., Koch, N. (eds.) ICWE 2011. LNCS, vol. 7059, pp. 49–61. Springer, Heidelberg (2012)

[3] `http://rave.apache.org`