Extending Web Standards-Based Widgets towards Inter-Widget Communication

Olexiy Chudnovskyy, Sebastian Müller, and Martin Gaedke

Chemnitz University of Technology
{olexiy.chudnovskyy,martin.gaedke}@cs.tu-chemnitz.de,
sebastian.mueller@s2011.tu-chemnitz.de

Abstract. In the last decade user interface mashups have gained much interest both in academia and in industry. Their development paradigm enables end users to develop applications without dealing with complexities of the underlying technologies by using so-called widgets. However, user interface mashups haven't reached their full potential yet. Widgets are currently implemented in an isolated manner hindering seamless user-mashup interaction due to the need of manual state synchronization between widgets. Even though mashup run-time environments often enable inter-widget communication, current widget implementations surprisingly make use of it seldom. In this paper, we present a generic approach to semi-automatically extend widgets with dedicated inter-widget communication functionality. Thus, facilitating many cumbersome tasks of the end users when combining different widgets to a single application.

1 Introduction

In the last decade plenty of data sources, Web services and user interfaces have been published on the Web, which led to new development paradigms focusing on reuse and composition [1]. One very promising approach in this context are user-interface mashups (UI mashups), which have the goal to enable end-users to develop their own solutions by simply combining functionalities at the user-interface level [2]. In contrast to data and service mashups, which require understanding of basic programming techniques, UI mashups support end-user development techniques like visual composition, immediate feedback, recommendation etc.

The building blocks for composition of UI mashups are widgets, which have the goal to hide the complexity of utilized data sources and Web services from end-users. UI mashup environments additionally provide messaging facilities, so that widgets can communicate their internal state among each other, allowing different widgets to act like one, and as a result improve the overall user experience [3]. For example, a widget, whose goal is to display a weather forecast for a given day, may adapt its display to a date chosen in a calendar widget. A map widget can adapt its focus to a new location, if it gets notified about the user's choice of a contact in the address book widget.

M. Grossniklaus and M. Wimmer (Eds.): ICWE 2012 Workshops, LNCS 7703, pp. 93–96, 2012.

Unfortunately, almost none of the currently available widgets on the Web support this inter-widget communication (IWC). In this paper, we present our approach to tackle the aforementioned problems. We show how stand-alone widgets can be extended towards a particular IWC technology. The rest of the paper is structured as following. Section 2 gives some background of inter-widget communication. In section 3 we present our framework to observe and extend existing widgets towards IWC-enabled ones. Finally, Section 4 concludes the paper and gives an outlook of our further research.

2 Background on Inter-Widget Communication

Inter-widget communication is a widely adopted aid both to improve user experience in UI mashups and to enable seamless execution of widget-based workflows. It is based on message transfer between isolated contexts, which can occur either in a centrally controlled manner (so called *orchestrated model*), decentralized self-organized manner (so called *choreographed model*) or decentralized inhibited manner (so called *hybrid model*) [4]. In our approach, we focus on the *choreographed* IWC model, where interactions between widgets are not centrally defined but occur in a distributed manner depending on events taking place inside the widgets. Either a user or a widget can raise a message to be emitted. It contains both type of the event occurred (e.g. city selection has changed) and further information about the event (e.g. the current city selection is "Chemnitz, Germany"). In the choreographed model the message is published on a dedicated event bus provided by the UI mashup environment. Other widgets may subscribe to a particular event type, so that they get notified when the corresponding message is published. The message is then passed to a widget-internal callback-routine, so that widgets can react to the event appropriately. We believe that though the choreographed IWC model requires syntactical and semantic compatibility between emitted messages, it is still best suited for end user development. As such, user don't need to explicitly configure the inter-widget communication and can re-use the IWC-functionality among different deployments.

3 Extending Widgets with IWC-Functionality

The extension of widgets is performed by deploying and modifying the corresponding widget packages in a dedicated "widget extension environment" (Figure 1). The extension environment performs a learning process, during which it collects data about the widget behavior and then extends the widgets with modules required for inter-widget communication. Some of the injected modules are responsible for publishing events to the global event bus; whereas the other ones subscribe to those events and replay certain actions if the events occur. During the learning process user can also specify how the event messages look like. To establish interoperability between sending and receiving widgets it is required to use the same vocabulary while describing outgoing and incoming messages of both widgets.

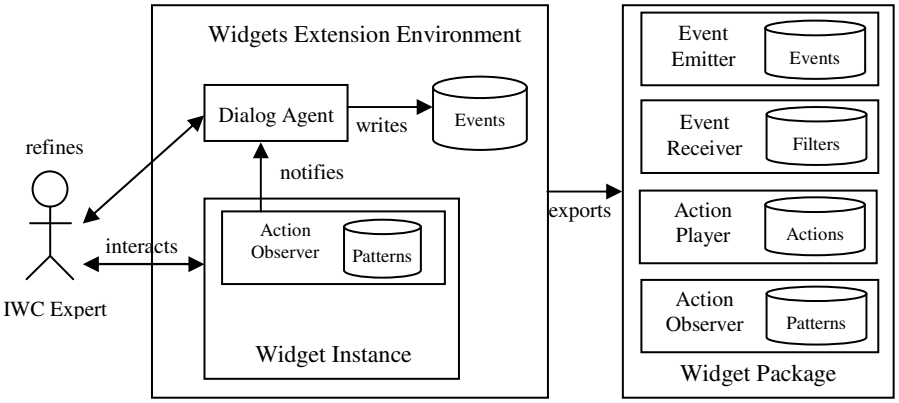

Fig. 1. Widget Extension Environment

The first step of the learning process foresees an extension of widgets towards emitting messages on internal state changes.

For this purpose a dedicated widget observer module is automatically embedded into the widget source code. The main goal of the module is to observe actions taking place in the widget and to detect pre-defined patterns from a given knowledge base. In the current implementation the observation is taking place by attaching dedicated event handlers to HTML input elements and accumulating data until some pattern is recognized (e.g. text input and subsequent form submission). After a pattern has been detected, the observer instructs the learning environment to clarify the performed action. A dedicated dialog with captured data is presented to the user, so that she can refine the semantics of the actions and annotate the corresponding parameters (Figure 2).

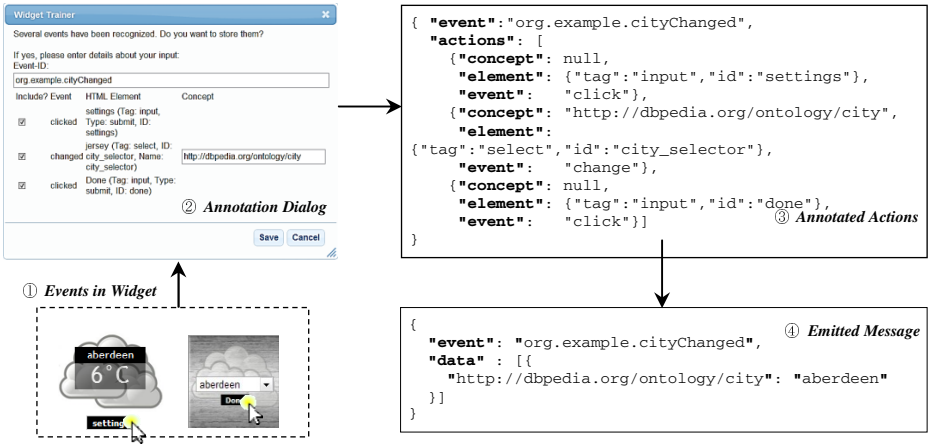

Fig. 2. Refining events to be emitted

The approach assumes that the user is familiar with concepts from choreographed IWC and can enrich the captured data with sufficient semantics. The refined event

data and corresponding occurrence pattern are then stored within a dedicated events repository. When the user decides to finish the training process, the learning environment automatically injects another module, the event emitter. The widget package is then exported as a new extended widget version. At run-time the action observer notifies the event emitter about detected actions, which converts them into the semantically enriched form and publishes to the global event bus.

As the second step towards IWC-enabled widgets, we inject modules being able to receive and process events published on the global event bus. For this purpose, we again use the observer module for recording user actions within the widget. In contrast to the prior approach, the recorded data is not used to define messages to be published, but to specify actions to be replayed if certain external events occur. Users can stop the recording process at any time and parameterize the recorded actions according to expected data in external events. To trigger and to repeat the recorded actions at run-time, the widgets extension environment injects two additional modules into the widget package – an event receiver and an action player. The event receiver module takes messages of the pre-defined type from the global event bus and passes them to the action player, which replays actions recorded during the learning process.

4 Conclusions and Outlook

In this paper we presented our approach to extend widgets towards IWC-enabled ones. We believe, this possibility will significantly improve the user experience within user interface mashups and will enable seamless workflow execution without need of manual state synchronization between involved components. We implemented our approach based on the W3C widget specification and Apache Wookie widget container, which was extended in order to support required observation and module injection features.

A screencast demonstrating capabilities of the widget extension environment can be found at http://vsr.cs.tu-chemnitz.de/demo/iwc-extension. Our future research will focus on more sophisticated learning techniques of user-widget interaction, making the IWC experience more comprehensive and efficient.

Acknowledgment. This work was supported by funds from the European Commission (project OMELETTE, contract no. 257635).

References

[1] Al Sarraj, W., De Troyer, O.: Web mashup makers for casual users. In: Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services, iiWAS 2010, p. 239 (2010)

[2] Daniel, F., Yu, J., Benatallah, B., Casati, F., Matera, M., Saint-Paul, R.: Understanding UI Integration: A Survey of Problems, Technologies, and Opportunities. IEEE Internet Computing 11(3), 59–66 (2007)

[3] Zuzak, I., Ivankovic, M.: A Classification Framework for Web Browser Cross-Context Communication. CoRR, vol. abs/1108.4 (2011)

[4] Wilson, S., Daniel, F., Jugel, U., Soi, S.: Orchestrated User Interface Mashups Using W3C Widgets. In: Harth, A., Koch, N. (eds.) ICWE 2011 Workshops. LNCS, vol. 7059, pp. 49–61. Springer, Heidelberg (2012)