Towards Real-time Collaboration in User Interface Mashups

Alexey Tschudnowsky, Michael Hertel, Fabian Wiedemann and Martin Gaedke
Department of Computer Science, Technische Universitt Chemnitz, Str. der Nationen 62, Chemnitz, Germany

Keywords: User Interface Mashup, Widgets, End-user Development.

Abstract: Recently, user interface mashups have become a popular approach for covering the long tail of end-user needs. The simple composition paradigm and rich tool support aim at enabling even non-programmers to aggregate data and application logic to solve situational business tasks. Despite of existing assistance mechanisms, both development and usage of UI mashups remain mostly a *single-user* activity. As a result, novices are usually left alone in case of configuration problems or unexpected errors. Furthermore, the work cannot be distributed efficiently to solve tasks faster. This paper presents an approach to make development of and interaction with mashups a collaborative process. We show, how one can extend today's composition platforms towards real-time collaboration and demonstrate the approach in the context of the open-source mashup platform Apache Rave.

1 INTRODUCTION

Dashboard-like applications (known also as user interface mashups or, shortly, UI mashups) have been widely adopted in enterprises (Hurwitz et al., 2005). They enable management of complex, heterogeneous and scattered information by end-users, so that situational problems can be solved faster and more efficiently (Cappiello et al., 2011). UI mashups consist of autonomous but cooperative visual components called widgets, which aggregate data from various sources and enable access to often used functionalities. Within a mashup widgets are able to synchronize their views using a dedicated messaging infrastructure. Though the development process is rather simple, novice users still lack initial knowledge on how particular mashup platform works and have little support in case of configuration problems. Furthermore, if working in teams, users can hardly parallelize their activities, as the majority of today's mashup tools are single-user applications.

The following scenario illustrates the problem. Alice and Bob, two company employees, plan a joint business trip to Vienna, Austria. To prepare the trip they want to collect information about their destination, perform various bookings and agree on the schedule. In contrast to Bob, Alice is already experienced in developing mashups, and, thus, she quickly constructs a dashboard, which aggregates and compares data from different Web sources. If working at the office, Alice explains to Bob capabilities of the composition platform and they use the resulting mashup to prepare the trip. However, the planning becomes much more challenging if the employees work remotely. Without appropriate collaboration facilities, Alice cannot efficiently share her view on a mashup and explain required configuration. For the same reason Bob can neither participate in the composition nor speed up the process.

We believe that real-time collaboration and communication facilities can efficiently support situations like this. However, equipping Web applications (and mashup platforms in particular) with collaboration facilities is challenging and requires careful design of synchronization algorithms, conflict resolution and awareness services (Heinrich et al., 2012b). The peculiarity of the UI mashup domain lies, additionally, in the fact that the aggregated components, i.e. widgets, can originate from third party developers and, thus, be highly heterogeneous. Furthermore, many mashup platforms host widgets in isolated security contexts (e.g. iframes), so that Same-Origin-Policy hinders direct access to their internal data or UI. While isolation helps to prevent possible leak of data, it represents a serious obstacle for synchronization of widgets across mashup boundaries.

This paper tackles the above challenges by deriving a reference architecture and guidelines to enable real-time collaboration in UI mashups. The main contributions are threefold: First, a terminology and conceptual approach to synchronize UI mashups is presented (Section 2 and 3); second, a reference architecture for integration of real-time collaboration facilities into mashup platforms is derived (Section 4); and finally, implementation of the approach in the context of an existing open-source mashup platform is demonstrated (Section 5). A review of related work is given in Section 6. Section 7 concludes the paper and gives an outlook on the planned evaluation and future research.

2 UI MASHUPS: STATE OF THE ART

In the following, we introduce terminology of UI mashups by adapting the definitions introduced by Wilson et.al. in (Wilson et al., 2012). The terminology is used for analysis, which artefacts are involved into real-time collaboration and helps to define synchronization algorithms for UI mashups.

Conceptually, a UI mashup is a tuple $m =< M,W,I >$ with:

- $M =< name, description >$ being a basic metadata of the mashup
- $W = \{w_i : w_i =< S_i, A_i, Pos_i, Dim_i >\}$ being a set of widgets w_i with relative position in a mashup $Pos_i =< left_i, top_i >$, dimensions $Dim_i =< width_i, height_i >$, internal state S_i, and inter-widget communication (IWC) affordances A_i. The internal state S_i describes, for example, current focus of a map widget or its current zoom level. Affordances A_i are specific to the concrete strategy used for IWC. In (Wilson et al., 2012) authors distinguish between choreographed, orchestrated or hybrid ones. In case of the choreographed and hybrid IWC strategies the affordances are defined as $A_i =< E_i, O_i >$ with $E_i = \{e_{il} : e_{il} =< name_{il}, topic_{il} >\}$ being a set of events generated by a widget and $O_i = \{o_{il} : o_{il} =< name_{il}, t_{il} >\}$ being a set of supported operations (Wilson et al., 2012). $t_{il} \in T$ are well-known communication channles (*topics*) defined by a topic ontology T. On occurance of internal event e_{il}, widget publishes a message on topic t_{il}. Other widgets, which support operations for the same topic, i.e. *subscribe* to t_{il}, are notified about the new message and execute their respective operations automatically.
- I being an IWC configuration of a mashup specific to the utilized IWC strategy. For hybrid IWC strategy it contains restrictions put by user on communication between a pair of widgets: $I = \{r_j : r_j =< w_{jsender}, w_{jreceiver}, t_j >\}$ with $w_{jsender}, w_{jreceiver} \in W$ and $t_j \in T$.

From the implementation point of view, a widget w is considered to be a standalone, autonomous Web application with its own user interface, application logic and persistence layer. The source code of a widget is usually packaged for distribution, e.g. in a dedicated ZIP archive (cf. W3C Web Apps[1]) or modularized XML file (cf. OpenSocial[2]). To be composed with others, a widget needs to be first instantiated and integrated into the hosting Web application. The state S of an aggregated widget is, conceptually, the same as the state of a traditional Web application (Lo et al., 2013). It reflects the widget's current DOM tree and states of Javascript objects. The hosting mashup acts as runtime environment for the aggregated widgets and operates among others a messaging infrastructure, which enables widgets to notify each other about internal state changes. Usually, widgets do not exchange data on low-level events (such as button clicks or text input), but rather on domain-specific ones such as form submissions or domain object selections. Depending on the utilized IWC strategy the infrastructure can either a global message bus, shared memory or registry of communication endpoints.

Lifecycle of a mashup can be roughly divided into a development and an execution phases. During a development phase mashup designer specifies mashup metadata, selects widgets for composition, configures their dimensions and initial state and specifies mashup IWC configuration. The result of the development phase is a specification, which can used by mashup runtime environment to instantiate the Web application. During the execution phase, mashup designer (or any other user) consumes information presented by the aggregated widgets and interacts with them to accomplish domain-specific tasks.

Despite the formal separation, development and execution phases of UI mashups are usually interweaved. The so called "live development" paradigm promotes smooth transition between the both in order to lower the barrier of mashup development(Aghaee and Pautasso, 2013). Many platforms and tools emerged, which enable runtime mashup reconfiguration. One reference architecture for such development and execution approach has been developed in the FP7 EU Project OMELETTE (OMELETTE Consortium, 2013) and its excerpt is presented in Figure 1.

[1]http://www.w3.org/TR/widgets/
[2]http://opensocial.github.io/spec/2.5.1/Core-Gadget.xml

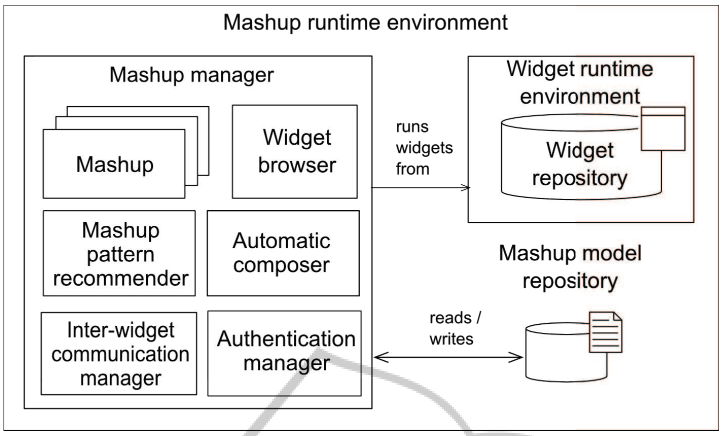

Figure 1: Reference architecture for UI mashups (terminology adjusted).

In the figure, the *Widget Runtime Environment* persists, instantiates and manages widgets. The *Mashup Runtime Environment* enables composition of widgets based on end-user-oriented programming techniques, e.g., using visual metaphors or natural language. Its submodules persist mashup models, manage user profiles and operate messaging infrastructure for IWC. *Widget Browser* enables manual discovery of widgets. To ease mashup development the runtime environment can be equipped with composition assistance services such as *Mashup Pattern Recommender* and *Automatic Composer* (Roy Chowdhury et al., 2013). Concrete Web implementations of the reference architecture are usually split into a server and a client sides. The server side is responsible for mashup persistence and management, while the client side renders a representation of a mashup, provides the IWC infrastructure and enables interaction with user.

3 REAL-TIME COLLABORATION IN UI MASHUPS

In the following, we analyze end-user and technological requirements on collaborative mashup development and usage. Afterwards, synchronization algorithms for both mashups and single widgets are presented.

3.1 Requirements Analysis

From a mashup designer's point of view, the main requirement is the ability to ”share” his view on a mashup with one or more collaborators and to interact with the mashup simultaneously. All changes to mashup structure and all interactions with widgets should be propagated and made visible to all collaborators in real-time. The awareness of mutual interactions should help to distribute work more efficiently and to minimize the number of possible conflicts. Finally, real-time communication facilities are desired to support coordination of actions during development and usage phases.

From the technical point of view, the main requirement for enabling real-time collaboration is the ability to keep mashup representations rendered in different contexts, i.e. browser instances, synchronized over time. Interactions with mashup and with widgets must be tracked, propagated and replayed in all contexts in real-time. Possible conflicts should be automatically resolved to keep all mashup representations in a consistent state. To support awareness of mutual presence and interactions, information about collaborators and about origin of remote changes should be visualized. Also interactions, which do not necessarily result in changes in mashup or widget UI (such as mouse movements or clicks), should be propagated. Finally, the platform should enable an on-demand transfer of text and multimedia data between collaborators with acceptable quality.

3.2 Real-Time Collaboration Facilities

According to the requirements above, integration of real-time collaboration facilities requires consideration of the following three aspects:

- Synchronization of mashups being rendered in different browser instances
- Tracking and visualization of active collaborators and their interactions
- Capturing, propagation and replay of media streams between collaborators

In the following, the three tasks and their conceptual realization are discussed in details.

3.3 Mashup Synchronization

First, we assume, that mashups representations being rendered in two browser instances of different users can be described as mashups from Section 2. We say, two mashup representations m_1 and m_2 are synchronized at some point of time t, if $m_1.M = m_2.M \wedge m_1.W = m_2.W \wedge m_1.I = m_2.I$. The real-time synchronization takes place, if changes to one mashup representation are applied or made visible to all others within some pre-defined time limit Δt. The real-time synchronization constraint creates an illusion of working on the same mashup, while in reality users interact only with local representations.

To keep all representations synchronized, we propose to apply the Operational Transformation (OT) algorithm (Ellis and Gibbs, 1989). Local modifications to metadata, widgets or IWC configuration of a mashup should be modelled as operations, which are observed and propagated to the master copy of the mashup representation. Each local mashup representation is assigned a revision number, which identifies a particular state of the mashup. Two mashup representations are considered to be synchronized if their revision numbers are equal. A conflict resolution engine should apply dedicated transformation functions to operations of different revisions, incorporate them into the master copy and propagate the transformed operations to all clients. Based on the mashup definition above the following operations have been identified (hybrid IWC strategy (Wilson et al., 2012) is assumed):

Op 1. $SetMashupName(name : string)$

Op 2. $SetMashupDescription(desc : string)$

Op 3. $CreateNewWidget()$

Op 4. $MoveWidget(w : Widget; left, top : int)$

Op 5. $ResizeWidget(w : Widget; width, height : int)$

Op 6. $DeleteWidget(w : Widget)$

Op 7. $AllowTopicBetweenWidgets(w_1, w_2 : Widget, topic : Topic)$

Op 8. $ForbidTopicBetweenWidgets(w_1, w_2 : Widget, topic : Topic)$

The transformation function for two operations (o_1, o_2) (o_1 having a higher revision) is defined as follows:

1. If $o_1 = o_2$ including all parameter with $o_1, o_2 \in \{Op1, Op2, Op4, Op5, Op7, Op8\}$, then the result of the tranformation function is $(NoOp, o_1)$ with $NoOp$ being the nothing affecting operation.
2. If o_1 and o_2 do not possess any equal parameters (meaning they do not affect the same artefact), then they are transformed to (o_2, o_1).
3. (o_1, o_2) with $o_1, o_2 \in \{Op4(w_i, ...), Op5(w_i, ...), Op7(w_i, ...), Op8(w_i, ...)\}$ except $(Op7(w_i, ...), Op8(w_i, ...))$ and $(Op8(w_i, ...), Op7(w_i, ...))$ is transformed to (o_2, o_1) as the affected widget properties are independent.
4. $(Op6(w_i), o)$ with $o \in (Op4(w_i, ...), Op5(w_i, ...), Op7(w_i, ...), Op8(w_i, ...)$ is transformed to $(NoOp, Op6(w_i))$, which avoids any operations on widgets, which have been deleted locally.
5. (o_1, o_2) with $o_1, o_2 \in (Op7, Op8)$ is transformed to $(NoOp, o_1)$.

Synchronization of widget states requires separate consideration. A complete synchronization can be achieved by exchange of complete DOM trees and Javascript objects of Web applications (Lo et al., 2013). However, if applied often, significant overhead especially in the context of complex HTML5 applications can be caused. Second, access both to widget DOM tree and Javascript variables is usually restricted for security reasons and, thus, generic change listeners and object modifiers would not work. To tackle the above challenges, we propose to use alternative synchronization mechanism for widgets. Widgets should reuse the existing IWC interface and notify their hosting environment about *domain-relevant state changes only*. The IWC interface is usually designed to bridge different security contexts by offering high level messaging operations. The mashup communication infrastructure should propagate the issued notifications to runtimes of other collaborators and invoke special IWC operations to apply the updates to corresponding widgets. The IWC operation to be invoked is the one, which has been assigned to a topic with an agreed naming convention (e.g., $original - topic - name$ concatenated with the suffix $-sync$). The operation (implemented by widget developer) should update widget UI and internal variables according to the incoming data.

To illustrate the synchronization of widget states, consider the example from Section 1. Every time Alice selects a new city on the map widget, the widget issues notifications about its internal state change (e.g., an IWC event on topic *city* and *Vienna* as payload). To display the same city in the Bob's map widget, the Alice's mashup runtime environment propagates the event to the Bob's one, which in turn notifies the local map widget about incoming update. The data is propagated on the topic *city-sync* and the Bob's map widget triggers a dedicated operation subscribed to this topic. It updates widget UI and internal objects to set the map focus on the new location.

The advantage of the method is that users can simultaneously interact with widgets. The drawback is, however, that they may put widgets into an inconsistent state. The reason is that events and operations affect only parts of the state, which might have interdependencies (e.g. selected city and country). To guarantee consistent widget state is in general challenging, as semantics of the events is domain-specific. Inconsistent states can be avoided on costs of simultaneous modification, e.g. by designing event parameters in the way, so that they reflect the most of the internal widget state. In best case, widgets propagate representations of the whole state, so that any eventual inconsistencies produced by local modifications are overwritten by incoming updates (last state update wins). Nevertheless, generic consistency preservation in widgets is still an open issue and should be considered in future research.

3.4 Presence and Awareness

As mentioned earlier, mashup runtime environments usually keep track of user profiles and mashup models. To take collaboration sessions with involved users and mashups into account, we propose to put the list of mashups shared to a particular user directly into his profile. While instantiating a certain mashup specification, runtime environment may check how many users have requested the same mashup and if they are still reading/writing the master representation. The check should result in a list of active collaborators, which can be displayed to all users to indicate presence information. The awareness of users about mutual activities should be established by visualizing their mouse movements and clicks to other users. One possibility to capture and display the mouse position is to track the identifier or XPath location of the element, where the cursor of a user is currently located and to propagate this information to other runtime environments for visualization. The same should be done for single and double mouse clicks. The method supports heterogeneous devices with different display as it doesn't stick to any absolute coordinates but rather to the DOM tree. The drawback is, however, that awareness inside of widgets in isolated security contexts cannot be established as such widgets hinder any access to internal mouse interactions.

3.5 Real-time Communication

For the communication of users during development and usage of mashups, we propose to utilize instant messaging, audio and video chat facilities. The three methods have been proven to be highly efficient on the Web and can be easily integrated into today's Web applications using latest HTML5 specifications. Communication endpoints correspond to browser instances of active collaborators. The functionalities are accessible on demand using a dedicated control panel in the mashup runtime environment.

4 REFERENCE ARCHITECTURE

The integration of the above facilities results in a new updated reference architecture illustrated in the Figure 2.

The common *Authentication Manager* now keeps track of shared mashups and takes care of collaboration sessions. The *Local State Manager* observes changes in mashup structure and listens to notifications issued by widgets. The data is forwarded to the *Global State Manager*, which maintains the master representation of the shared mashup, performs eventual conflict resolution (OT transformations as described in section 3.3) and propagates the updates to other *Local State Managers*. On receipt, *Local State Managers* apply the changes to local mashups and invoke dedicated IWC primitives to update states of their widgets. Interactions, which affect neither mashup nor states of widgets, such as mouse movements, clicks or presence changes, are observed and visualized in all mashup instances by a dedicated *Awareness Manager*. Finally, *RTC Manager* provides instant messaging, audio and video chat services to collaborating participants. Depending on the direction of communication (uni- or broadcast), messages and multimedia data are transferred to one or more mashup runtime environments.

5 IMPLEMENTATION

To demonstrate the proposed concepts we extended the open source UI mashup platform Apache Rave[3] towards real-time collaboration. Apache Rave enables aggregation of W3C and OpenSocial widgets on one canvas and provides IWC infrastructure based on OpenAjaxHub[4] (choreographed model). Widgets' source code is deployed to Apache Wookie[5],

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

[4]http://www.openajax.org/member/wiki/OpenAjax_Hub_2.0_Specification

[5]http://wookie.apache.org

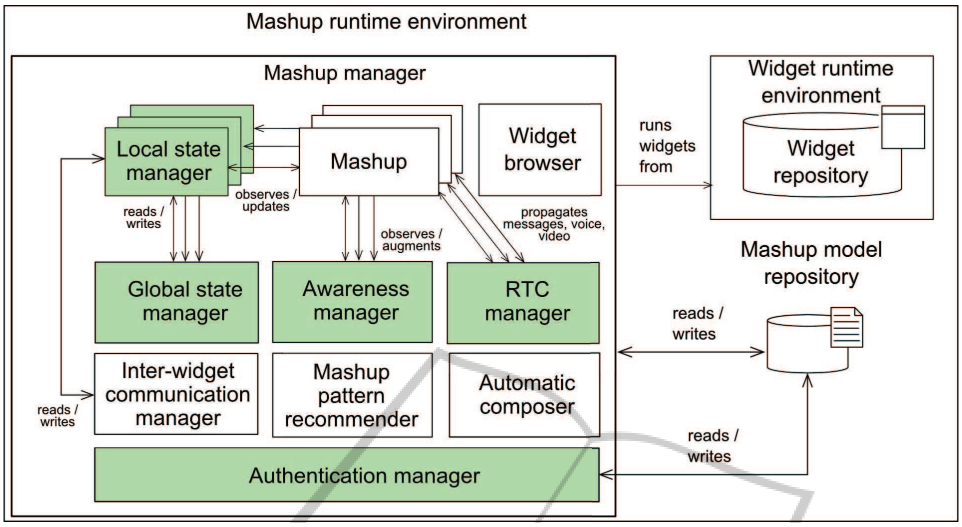

Figure 2: Reference architecture of UI mashups with integrated real-time collaboration facilities.

which enables their instantiation and provides various runtime services such as persistence or proxy facilities. Widgets can autonomously publish and subscribe messages on a shared message bus. With activated IWC configuration extension (Tschudnowsky et al., 2014), mashup designers are able to adjust IWC using widget isolations and topic blockades. In its current version, Apache Rave provides only rudimental collaboration facilities - mashups can be shared among users, but no real-time synchronization, awareness or communication services are provided. Updates to shared mashup instances become visible only on the next page reload (mashup specification is used as a master model of mashups). No conflict resolution mechanism exists.

The platform has been extended towards the reference architecture from Section 4 (cf. Figure 3).

Awareness and RTC Manager are implemented using the TogetherJS[6] framework, which also acts as messaging infrastructure for global state management. TogetherJS consists of a JavaScript library and a messaging server, called *TogetherJS Hub*. The decision to use this particular framework was motivated by the fact, that the framework offered infrastructure, which could be easily reused and extended towards the peculiarities of the UI mashup domain.

The client library has been integrated into the front-end of Apache Rave. Upon loading, all local mashups connect to the TogetherJS hub and join a common *session*, which is used to group collaborators and to broadcast update notifications. Client library enables message exchange using publish-subscribe strategy. Local State Managers are implemented as a set of callback routines, which are invoked on changes to mashup structure, layout and IWC configuration. These changes and IWC notifications of widgets on internal updates are recorded and are propagated to the TogetherJS hub, which routes them to Local State Managers of shared instances. The updates are then either applied directly to corresponding local models or passed to IWC state-update operations of widgets (cf. Section 3.3). Mouse movements, clicks and presence changes are tracked and visualized. Real-time communication facilities include text and audio chat implemented using WebSockets[7] and WebRTC[8] protocols respectively.

Online Demonstration. A demonstration of the proposed facilities is available at http://vsr.cs.tu-chemnitz.de/demos/collaborative-ui-mashups.

6 RELATED WORK

A number of frameworks emerged to support integration of real-time collaboration features into generic Web applications. The frameworks implement operational transformation (OT) algorithms on various data structures (text, XML, JSON objects etc.) and support concurrency control in real-time. ShareJS[9], OpenCoweb[10] and Apache Wave[11] are some of the examples of such frameworks. However, peculiarities of application domain should be considered during their integration. A generic approach to transform

[6]https://togetherjs.com

[7]http://dev.w3.org/2011/webrtc/editor/webrtc.html

[8]https://tools.ietf.org/html/rfc6455

[9]http://sharejs.org

[10]http://opencoweb.org

[11]http://incubator.apache.org/wave

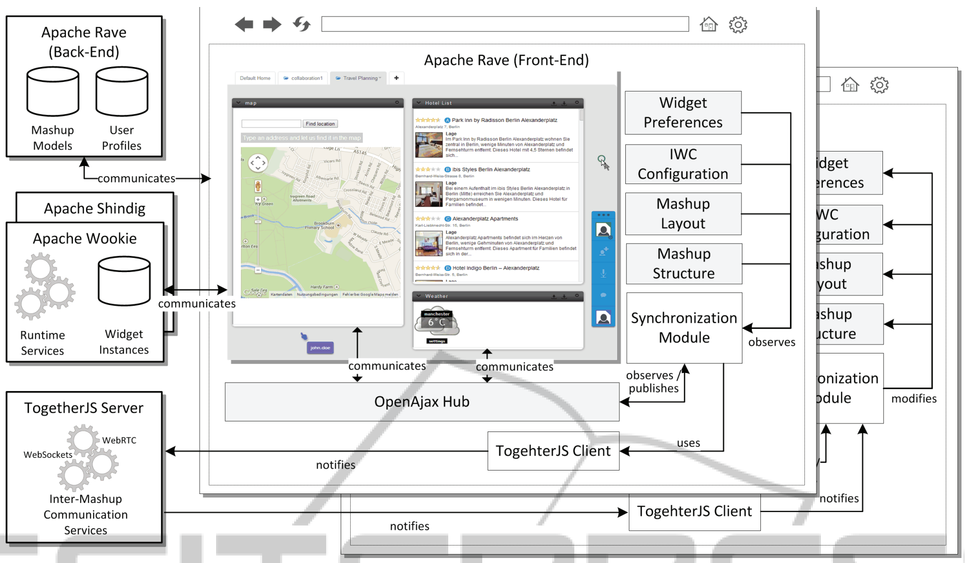

Figure 3: Integration of real-time collaboration facilities into Apache Rave.

single-user Web applications into multi-user ones is presented in (Heinrich et al., 2012b). Authors propose to synchronize multiple copies of Web applications on the DOM tree level. In (Heinrich et al., 2012a) the approach is enriched with awareness widgets such as telepointer or participant list. Though, the presented solution significantly simplifies implementation of collaborative applications, it doesn't consider synchronization of application-specific Javascript objects and widget being executed in isolated environments. A few commercial and open-source projects integrated real-time collaboration facilities into the respective mashup platforms. Adobe Genesis[12] is one commercial solution based on Adobe Air, which enables composition of widgets (called tiles) and sharing of the resulting view with other users. Liferay[13] is an open-source Web-based portal software that can be extended towards real-time collaboration using the commercial plugin OpenScape Web Collaboration[14]. Due to the lack of information on the internal implementation, it is difficult to compare or reuse utilized solutions to the presented problems.

Several research projects attempted integration of real-time collaboration facilities into UI platforms. PEUDOM (Matera et al., 2013) is an end-user-oriented UI mashup platform, which enables collaborative composition of widgets. Shared mashups are synchronized in real-time - both their structure and widget states. Authors apply a similar synchronization approach based on explicit widget state exposure. Similar approach for widgets described on an abstract level has been considered by (Blichmann et al., 2013).

Fine-grained access control in mashups with real-time collaboration support has been considered by Sire et.al. in (Sire et al., 2009). Authors develop a dedicated specification language and discuss various use cases. The work doesn't focus on engineering aspects of collaborative mashup platforms, but could be used in conjunction with the presented work.

In contrast to existing projects, this paper uses IWC as a mechanism for state observation, propagation and replay. If widgets support IWC, than the notifications on state changes can be efficiently reused. The interface requires only minimal changes (new IWC operations) to incorporate those updates into widgets of remote mashups. Additionally, this paper considers the problem of keeping mashups in a consistent state, which arises if users interact with a mashup simultaneously.

7 CONCLUSIONS AND OUTLOOK

Collaboration software has been widely applied within enterprises and on the Web. However, only little research can be found within UI mashup domain. Peculiarities of Web mashups require special considerations of involved data structures and synchronization of heterogeneous and secured components. This paper presented a systematic approach to integrate real-time collaboration facilities into UI mashup plat-

[12]http://helpx.adobe.com/genesis/topics/getting-started.html
[13]www.liferay.com
[14]http://www.unify.com/us/products-services/unified-communications/applications/web-collaboration.aspx

forms. A conceptual approach including mashup and widget synchronization algorithms, a reference architecture and an example implementation were given. One lesson learned, is that though current technologies and protocols significantly simplify implementation of collaborative Web applications, a careful design is still needed if applied to a concrete domain. By now, we haven't found a generic automatic solution for syncronization of widgets being executed in secured contextes. As a result widget developers are required to extend the IWC interface manually. At least for controlled environments, where source code of widgets is available, we are exploring application of automatic source code transformation towards the extended IWC interface.

A detailed evaluation of the integrated real-time collaboration facilities is planned. A user study and performance evaluations should help to assess usability and efficiency of the current solution. The performance evaluations should provide data on, how fast updates are exchanged between mashups and how many conflicts emerge in practice. The usability study should unveil, how comfortable users are with the integrated mechanisms, and if the current tools sufficiently support knowledge exchange and work distribution.

REFERENCES

Aghaee, S. and Pautasso, C. (2013). Live mashup tools: Challenges and opportunities. San Francisco, CA, USA.

Blichmann, G., Radeck, C., and Meiner, K. (2013). Enabling end users to build situational collaborative mashups at runtime. In *8th International Conference on Internet and Web Applications and Services (ICIW2013)*.

Cappiello, C., Daniel, F., Matera, M., Picozzi, M., and Weiss, M. (2011). Enabling end user development through mashups: requirements, abstractions and innovation toolkits. In *Proceedings of the Third international conference on End-user development*, IS-EUD'11, pages 9–24, Berlin, Heidelberg. Springer-Verlag.

Ellis, C. A. and Gibbs, S. J. (1989). Concurrency control in groupware systems. *SIGMOD Rec.*, 18(2):399–407.

Heinrich, M., Grüneberger, F. J., Springer, T., and Gaedke, M. (2012a). Reusable awareness widgets for collaborative web applications - a non-invasive approach. In *ICWE*, pages 1–15.

Heinrich, M., Lehmann, F., Springer, T., and Gaedke, M. (2012b). Exploiting single-user web applications for shared editing: a generic transformation approach. In Mille, A., Gandon, F. L., Misselis, J., Rabinovich, M., and Staab, S., editors, *WWW*, pages 1057–1066. ACM.

Hurwitz, J., Halper, F., and Kaufman, M. (2005). Dashboardsenabling insight and action. Technical report, Hurwitz and Associates.

Lo, J. T. K., Wohlstadter, E., and Mesbah, A. (2013). Imagen: Runtime migration of browser sessions for javascript web applications. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 815–826, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

Matera, M., Picozzi, M., Pini, M., and Tonazzo, M. (2013). Peudom: A mashup platform for the end user development of common information spaces. In Daniel, F., Dolog, P., and Li, Q., editors, *Web Engineering*, volume 7977 of *Lecture Notes in Computer Science*, pages 494–497. Springer Berlin Heidelberg.

OMELETTE Consortium (2013). D2.3 - final specification of mashup description language and telco mashup architecture. Public deliverable, The OMELETTE Project (FP7/2010-2013 grant agreement n 257635).

Roy Chowdhury, S., Chudnovskyy, O., Niederhausen, M., Pietschmann, S., Sharples, P., Daniel, F., and Gaedke, M. (2013). Complementary assistance mechanisms for end user mashup composition. In *Proceedings of the 22Nd International Conference on World Wide Web Companion*, WWW '13 Companion, pages 269–272, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

Sire, S., Palmr, M., Bogdanov, E., and Gillet, D. (2009). Towards collaborative portable web spaces. In *Second International Workshop on Mashup Personal Learning Environments (MUPPLE09)*, pages 80–88.

Tschudnowsky, A., Pietschmann, S., Niederhausen, M., and Gaedke, M. (2014). Towards awareness and control in choreographed user interface mashups. In *Proceedings of the 23nd International Conference on World Wide Web companion (to appear)*, WWW 14 Companion.

Wilson, S., Daniel, F., Jugel, U., and Soi, S. (2012). Orchestrated user interface mashups using w3c widgets. In *Proceedings of the 11th International Vonference on Web Engineering*, ICWE'11, pages 49–61, Berlin, Heidelberg. Springer-Verlag.