MultiMasher: A Visual Tool for Multi-device Mashups

Maria Husmann, Michael Nebeling, and Moira C. Norrie

Institute of Information Systems, ETH Zurich CH-8092 Zurich, Switzerland
{husmann,nebeling,norrie}@inf.ethz.ch

Abstract. The proliferation of a wide range of computing devices from tablets to large displays has created many situations where we no longer use a single device only. Rather, multiple devices are commonly used together to achieve a task. However, there is still little tool support for such scenarios in which different devices need to be combined to control an interface. Our goal is to enable multiple devices to view and interact with multiple web resources in a coordinated manner based on our new idea of multi-device mashups. In this paper, we present a first, visual tool for mashing up devices to access web sites, and discuss how we addressed the challenges as well as interesting issues for further research.

1 Introduction

Nowadays, it is not uncommon for a person to possess multiple computing devices such as a mobile phone, tablet, desktop and laptop computer. At the same time, there are devices that are shared by groups of users such as interactive tabletops or large screens. For example, in a group meeting there could be a mobile phone for each participant, as well as tablets, a laptop computer for the person leading the discussion and a large screen which is used to share information with participants. In order to fully profit from such kinds of ecosystems, it is desirable to use the devices in combination, rather than using each device in isolation. However, this is currently often not possible because most applications are designed for a single device. Moreover, migration between devices requires additional mechanisms and infrastructure as well as modifications to the applications themselves [1].

While recent research in the area of distributed user interfaces has focused on developing frameworks and tools [2,3] to systematically support the distribution and migration of applications across devices, most solutions are usually limited to considering only a single application [4]. On the other hand, there is also a great body of research on web mashups that combine content, presentation and functionality from different web-based sources, with the aim of creating new applications from existing resources. However, research on mashups has so far focused on combining multiple existing web sites and services on a single device [5], whereas our goal is, not only to combine multiple web resources to create a new application, but also to support mashing up multiple devices so that

Q.Z. Sheng and J. Kjeldskov (Eds.): ICWE 2013 Workshops, LNCS 8295, pp. 27–38, 2013.

applications can be distributed between, and accessed from, different devices at the same time. We define a multi-device mashup as a web application that reuses content, presentation, and functionality provided by other web pages and that is distributed among multiple cooperating devices.

Our new idea of such multi-device mashups introduces a number of interesting challenges. Some of these have partly been addressed by the respective research communities in distributed interfaces and mashups, but they need to be combined, adapted and extended to support the design and development of mashup applications that involve multiple devices. For example, while the mashup community has come up with ways to introduce interactions between components that originate from different sources [6], these need to be expanded to support interactions across devices. Instead of each device accessing web content in isolation, in a multi-device mashup, access needs to be coordinated [4]. For example, a selection of a list item on one device may trigger an update both on the current and connected devices. Moreover, mashups traditionally assume a single device and user, but multi-device mashups may involve multiple devices as well as multiple users when devices are shared between users.

To address these challenges and experiment with possible solutions, we developed *MultiMasher*—a visual tool for rapidly designing multi-device mashups. We motivate the need for such a tool by presenting a scenario for a multi-device mashup in Section 2. In Section 3, we then give an overview of MultiMasher, followed by architecture and implementation details in Section 4. Section 5 picks up on some of the issues and discusses them in more detail, while Section 6 gives concluding remarks and an overview of opportunities for further research.

2 Scenario

To illustrate the usage of a multi-device mashup, we consider the following scenario also depicted in Fig. 1: two users, Alex and Bill, are planning a mountain biking trip together at home in their living room. They need to decide on a route and find out how they can get to the starting point by train. Alex has a smartphone, Bill a tablet, and additionally they share their TV for a larger display area. Alex and Bill have split their tasks in the following way: Bill is browsing a route repository such as SwitzerlandMobility[1] and Alex is querying the connections on the web site of the local railway company. When Bill inspects a route, the starting point of the route is sent to Alex' smartphone where she can browse the available railway connections, while Bill checks whether the route suits their criteria for length, difficulty, and scenery. Some starting places may take too long to reach, others may require too many changes of trains, which is a hassle when travelling with bikes. The resulting railway connections of Alex' query are displayed on her smartphone in a list. She can select a connection for which the details are displayed on the large screen, where also Bill's information about the suggested route are visible. If Alex enters a new time on her mobile phone or selects a new connection, the detailed view is updated also on the large screen.

[1] `http://www.mountainbikeland.ch`

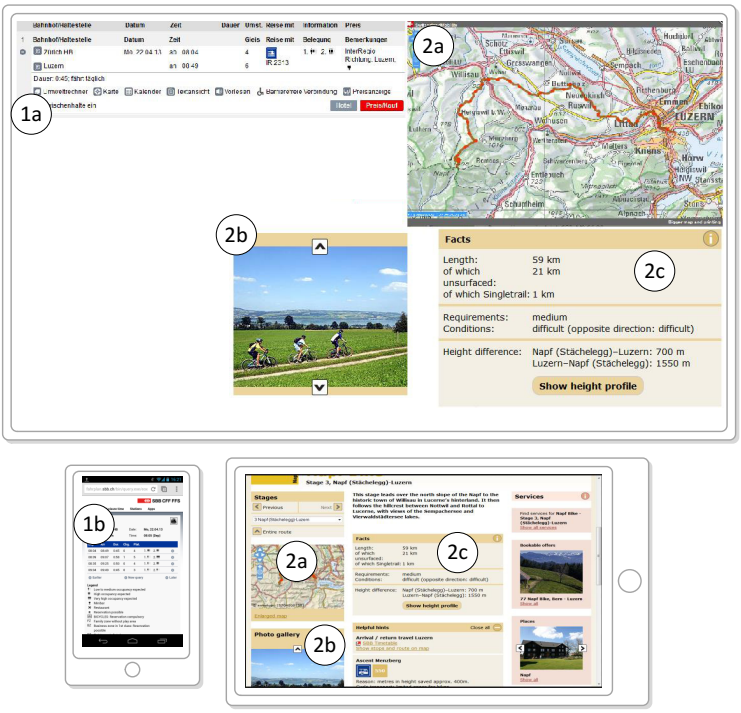

Fig. 1. A scenario using a shared screen, tablet and mobile phone for planning a mountain biking trip. 1a and 1b are elements from the railway website. 1b displays a list of possible railway journeys to a given destination. Selecting a journey updates the detail view in 1a. 2a to 2c are taken from the route repository website. Interaction with one of these elements on the tablet also updates the shared screen and vice versa. The destination for the query in 1b is taken from 2c.

Together they discuss whether the currently visible trip is a good match. During the discussion, they can also interact with the content on the large screen. If they are not happy with the current trip, Bill starts looking for another route.

3 MultiMasher

As a first step, we created MultiMasher, a visual tool that allows mashing up devices for accessing web content of multiple sources. The goal is to build a tool designed to facilitate the quick and easy creation of multi-device mashups, such as the one described in the scenario, through a direct manipulation interface, without the need for modelling or programming. MultiMasher is aimed at both developers and non-technical end-users to assist them in the rapid prototyping of distributed interfaces and development of multi-device mashups. With MultiMasher, users should be able to, not only create multi-device mashups for the devices at hand, but also decide how they should adapt to a different set of

devices. At a later stage, the mashups created through MultiMasher could also be used as a basis for complex mashup applications that will finally be deployed.

To create a mashup, a user needs to connect the devices involved to the MultiMasher server. This is done by loading the MultiMasher tool in the browser, which automatically connects the device to the server. More devices can be added at any time. Users can then load the web sites that they want to mashup, select UI elements on that web site, and distribute these to the connected devices. This can be done on any of the connected devices and is not restricted to the device that was initially used to start the distribution. For example, in our scenario, Bill could start the mashup on his tablet and Alex could join on her smartphone designing the parts concerning the railway information. Also, the user can save mashups and continue adapting them at a later point in time. In our scenario, Alex could decide later on to not only show the details of the selected railway connection on the large screen, but also the complete list of available connections. Thus, mashups can be created in an iterative manner in that existing mashups can be reused to form the basis of a new one.

MultiMasher distinguishes the following two modes: In the *design mode*,the users create a multi-device mashup. In the *distribution mode*, a multi-device mashup is active on the devices and the users interact with it.

MultiMasher includes a toolbar that can be loaded into existing web sites. The toolbar exposes the functionality of MultiMasher to the user. By default, the toolbar is hidden except for a small button which enables the design mode and slides in the whole toolbar (Fig. 2). Thus, the toolbar does not take up any screen space, which is rare on some devices, when it is not in use. When in design mode, the user can select UI elements and send them to connected devices, where the elements will be mashed up with the content that has been sent to the device previously.

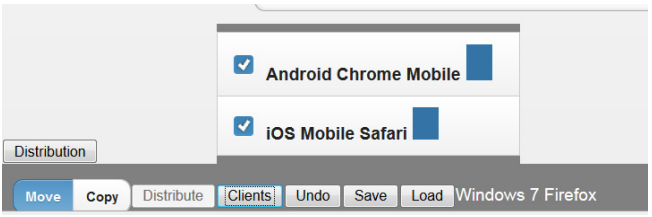

Fig. 2. The MultiMasher toolbar. The name of the local client is displayed on the right. The clients available for a mashup can be selected from a list (centre). On the left, the distribution mode can be selected.

When a user is creating a mashup, the present client devices are offered for selection (Fig. 2). In a list, each device is a represented visually. For every device, a name is generated automatically, making use of the operating system and browser information. This name can be changed by the user to a name of their own choice. Additionally, there is a visual representation of the available screen space in pixels for each device, which helps the users to map the physical devices

to the representations. We imagine that this could also be done in other ways, e.g. by mapping each device to a colour which would be displayed on the actual device and in the representation. Each device can be selected for participation in the mashup. The list of devices is updated as devices join or leave. When a new device joins, it can easily be added to an ongoing mashup. Therefore, it is not required that all devices are present when the user starts the process of creating a mashup. The toolbar also provides an undo functionality that causes the device to leave the current distribution and restores the original web site.

When the design mode is activated through the toolbar, the user can select UI elements by clicking on them or touching them on touch devices. This direct selection method should make it easy also for non-technical users to choose elements, as they do not need to know the underlying structure of the web site. MultiMasher provides visual feedback for selected elements (Fig. 3) by highlighting them. Any number of elements can be selected simultaneously. When at least one element is selected, the user can send it to connected devices, by pressing the distribute button in the toolbar. The selected elements will then be sent to the client devices selected in the device list. MultiMasher supports two modes for distributing UI elements. The first mode, move, sends the element to the target devices without keeping a copy on the source device. In contrast, the second, copy, sends the UI element to the target devices and retains a copy on the source device. In our scenario, the details for a railway journey have been moved from the smartphone to the large screen. They are no longer visible on the smartphone, but only on the large screen. The gallery and the facts for the mountain biking trip have been copied from the tablet to the large screen. They are present on both devices.

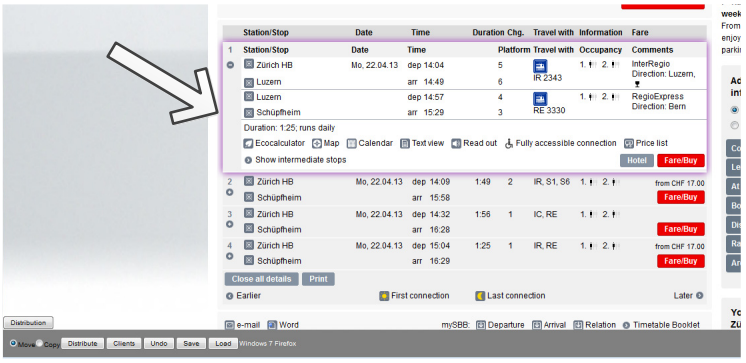

Fig. 3. Selecting UI elements in distribution mode. Coloured outlines provide visual feedback for selected elements.

One of the biggest challenges in MultiMasher was to synchronize interactions of a web site as the UI elements are distributed across devices. MultiMasher provides a first partial solution to synchronizing state by replaying interactions on each device. For example, if a menu allows the selection of content, the menu

and the content area can be on two separate devices, thus, leveraging the differing capabilities of each device, such as input modality and display size. Selecting a menu item on the first device will update the content on the second device. In our scenario, selecting a railway connection on the mobile phone, will update the corresponding detail view on the large screen. UI elements that are copied on multiple devices are also synchronized by default. For example in the scenario, a photo gallery is copied on the shared screen as well as on the tablet. As the two are synchronized, when the user navigates through the photos on the tablet, the gallery on the shared screen updates also.

In the future, MultiMasher should also allow users to define interactions across web sites by connecting elements from multiple sources. In our scenario, the starting location of the bike tour is connected to the destination input in the timetable of the railway company. As the user on the tablet navigates among possible bike tours, the user on the mobile phone gets an updated list of connections to the destination and can change query parameters such as the date, which are independent of the bike tour.

MultiMasher allows the user to save created mashups and load them at a later point in time. Once a mashup is loaded, it can be adapted by adding new devices, redistributing elements or adding new web sources. For example, in our scenario, a third user, Chris, could show up with his tablet and would like to search bike tours in parallel to Bill. His device would need to be integrated. Chris may also know another good source for bike tours and would like to show information from this new source on the large screen to Alex and Bill.

Also, multiple mashups can be merged. For example, if two mobile phones are participating in one mashup and a tablet is connected to a shared screen in another mashup, these two mashups can be combined. As a result, all four devices will be connected and interactions on a mobile phone may now update content on the shared screen. In our scenario, there could be an existing mashup of timetables on two mobile phones, one showing the list of connections and the other the details. Another existing mashup could combine the large screen and the tablet for viewing bike trips. The two can be merged by distributing the detail view from the mobile to the large screen in order to obtain the setup described in the scenario.

Saving, loading, and merging support an iterative and explorative work process that developers may employ when creating prototypes. Users can create parallel versions and merge them later on. On the other hand, non-technical users can load mashups created by experienced developers and do minor changes, such as adding a new device.

Saved multi-device mashups can be previewed in MultiMasher before being loaded on the connected devices. The preview tool (Fig. 4) lists all stored mashups that include the given device. When a mashup is selected, a preview of all devices participating is displayed. Each device is presented in a miniature version. Thus, the user can quickly see what devices are involved and how the content is distributed across the devices.

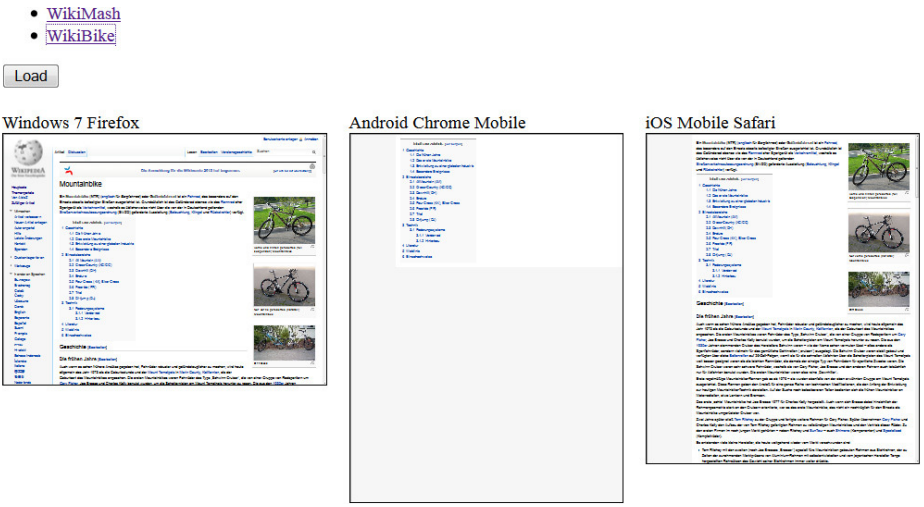

Fig. 4. Previewing mashups in MultiMasher. The devices are scaled according to available space in pixels.

4 Architecture and Implementation

In this section, we describe the overall architecture of MultiMasher and then give implementation details. Figure 5 illustrates the client-server architecture of MultiMasher. The server coordinates the communication between client devices and stores created mashups. The device manager informs clients of devices joining or leaving. It extracts operating system and browser information, which are used for device name generation.

The persistence manager stores mashups and device information in a database and retrieves them when an existing mashup is requested for loading. When a mashup is active, the server-side event manager receives events from the client-side event managers and forwards them to all clients involved in the mashup. During the process of creation and adaptation of a mashup, the selection engine in the server receives selections of UI elements from source clients that need to be displayed on target clients in the mashup. The selection engine merges these selections with previous selections for the target clients and notifies the presentation engine in the clients about the updates. The mashup manager handles the creation and adaptation of mashups. It receives changes to mashups from the selection engine and integrates them into the existing mashups. The mashup manager handles the merging of multiple mashups and propagates changes in mashups to the persistence manager. On the client side, an event engine forwards local events to the server and replays remote events locally. With this mechanism, the event engine ensures a consistent state across multiple devices. When a user interacts on source devices, the interaction triggers an event, which will be replayed on all devices in the mashup, thus simulating the interaction and causing the same actions on all devices. The selection engine on the client side

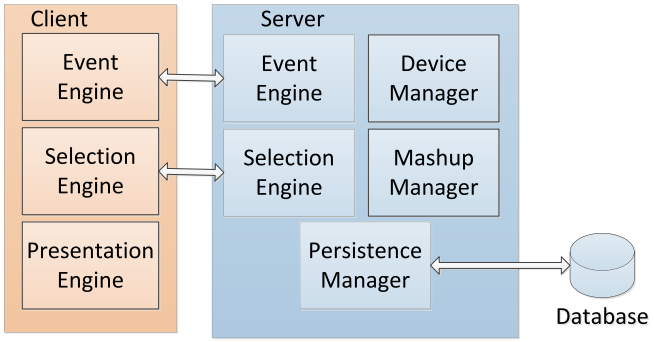

Fig. 5. MultiMasher architecture

allows the user to select UI elements, visualises current selections, and reports them to the server-side selection engine. The presentation engine processes selections from the server and adapts the UI so that only the requested elements are visible. All clients load a full version of the involved web site, but only show the selected elements and their parents. All elements that are not needed are simply hidden. In the future, the presentation engine could be extended with layout information for the visible elements.

The MultiMasher server is implemented in jQuery[2] on top of the Node.js[3] platform. Persistence is achieved with a MySQL database where information about mashups and client devices is stored. As the server needs to propagate events from a source device to all other devices in a mashup, a mechanism for server push is needed. We decided to use the socket.io[4] library which is integrated as a Javascript library on the client side and as a Node.js module on the server side. Socket.io transparently provides a server push mechanism using websockets, if supported by the browser, otherwise it employs fallback mechanisms such as longpolling. Socket.io informs the device manager when a device disconnects and, thus, the device manager is constantly aware of the connected devices.

On the client side, MultiMasher consists of a Javascript library that can be integrated into existing web sites. The integration can be achieved by modifying the HTML file of the web site or via a browser plugin such as Greasemonkey[5].

When the MultiMasher library is loaded into a web site, it injects a toolbar which allows the activation of a design mode in which UI elements can be selected. When in design mode, a transparent overlay is added over the web site. Thus all clicks or touches on the UI are captured by the overlay. This mechanism prevents clicks from activating actions on the web sites, such as following a link. From the click location on the overlay, we calculate the underlying UI element with the elementFromPoint Javascript function on the document node.

[2] `http://www.jquery.com/`
[3] `http://www.nodejs.org`
[4] `http://www.socket.io`
[5] `http://www.greasespot.net`

As we use id attributes for identification, we find the closest parent (including the selected element) in the DOM tree that has an id attribute using jQuery.

When a user designs a mashup, the client sends a list of devices and the selected UI elements to the server. The server notifies the devices of their participation in the mashup and propagates the selection. All elements that are hidden by MultiMasher are marked by a class, so the system can distinguish between elements that have been hidden by the web site itself and those by MultiMasher. Upon activation of a mashup, MultiMasher adds event listeners for DOM events such as click or change. When an event is detected, it is sent to the server, which forwards it to all devices participating in the mashup. When an event is received from the server, it is replayed in all other clients, thus causing the same reaction on all devices. As UI elements are only hidden but not removed from the DOM, events on them can still be triggered. A flag distinguishes local events from remote events. Thus, endless event cycles can be prevented by only sending local events to the server. The approach of replaying events is very simple, but it also has some limitations. In cases where an event triggers an update to the original web server of the source, the update would be executed multiple times. For events that only cause queries, the approach works fine.

5 Discussion and Related Work

Our work on MultiMasher builds on two, so far isolated, research streams on distributed interfaces and web mashups. Existing research in the area of distributed user interfaces has focused on using one existing interface and distributing it across multiple devices. On the other hand, research on mashups has focused on combining multiple existing web sites and services, but on a single device. We aim at the rapid creation of distributed interfaces by reusing parts from existing web sites, thus creating mashups that can span more than one device.

Recently, there has been increased interest in the systematic development of distributed user interfaces [4]. Many of the proposed approaches build from the research on model-based user interfaces that require a specific set of models organised into different abstraction layers [7] and user interface description languages. For example, the work presented in [3] builds on top of the UsiXML langugage and also the MARIA XML language has recently been extended to support distributed user interfaces [1]. When it comes to the implementation, the proposed solutions either require special cross-platform, peer-to-peer toolkits for general graphical user interfaces [8] or use HTML proxy-based techniques in the case of web applications [9]. Our goal was to provide a flexible tool that does not require specific models, languages and protocols and that is compatible with common web interface implementations based on HTML, CSS and JavaScript.

Common to most mashup tools such as Potluck [10], d.mix [11], Firecrow [12] and DashMash [13] is that they have been designed with non-technical end-users in mind. The general goal is to simplify the creation of web mashups based on visual tools for data aggregation and integration, graphical extraction and composition of various web resources and example code. Some of the latest

tools can even directly assist end-users in the mashup development process. For example, DashMash [13] can recommend other services that could also be useful in the current design context. However, with most of these approaches, it is not clear who defines and configures available services and how they can scale to supporting the development of complex web applications. An exception is MashArt [5,6] which is a platform designed to support web information system development in the form of mashups. MashArt targets advanced users with the goal of enabling them to create their own applications through the composition of interface, application and data components. The focus is on supporting the integration of existing web services and presentation components using an event-based paradigm so that components can react to events of other components.

While the component-based approach is generally promising, supporting reuse of web components that were originally created for different web sites is a complex issue for which different techniques have been proposed. For example, [14] presents a paradigm they call distributed orchestration which allows integrating web services and interfaces based on workflow and business process modelling techniques. While this approach enables inter-component communication, it is unclear how it supports mashup applications created for multiple users. In contrast, [15] presents an approach that focuses on adding awareness widgets to applications, which could in principle provide a basis for multiuser-aware mashups, but only considers one application and device. We regard our work on MultiMasher as a combination and extension of these approaches in that we support reuse of different web application components and allow also non-experienced users to distribute them across devices.

In this paper, we have presented a first implementation of MultiMasher that still has some limitations and will be the subject of both technical and user evaluations in the future. For example, we have tested MultiMasher on several existing web sites such as Wikipedia and achieved good results, but there are still open issues when the propagation of interactions generates multiple updates to a database, such as when a comment is posted using one device and the submit button also triggered on all connected devices, or when a web site requires user authentication and the mashup is in fact to be used by more than one user. For example, Twitter and Facebook could not be easily integrated in a mashup with MultiMasher. In contrast, we have achieved good results with web sites which present content without requiring an authenticated user, such as blogs, news, and reviews web sites.

At this stage, MultiMasher only allows users to design mashups for devices that are actually connected to the system at design time. We are currently working on a more flexible solution that allows designing for an unknown set of devices at run-time. For example, a mashup could initially be designed for two tablets. If, at a later stage, a mobile phone is used instead of one of the tablets, the phone could take over that role in the mashup. MultiMasher could analyse the available devices and their characteristics and offer this information to the user. In addition to allowing users to rearrange the mashup elements for the new device, the system could automatically scale the mashup according

to the difference in screen size, or otherwise adjust the layout, to facilitate the adaptation to different device capabilities and input modalities.

Moreover, changes may not only occur in the set of devices, but also in the web sites involved in the mashups. Therefore, additional mechanisms are required for MultiMasher to handle changes in the structure of web sites. Similar to other solutions, our selection method is currently restricted to web page elements that have an id attribute. The solution presented in [2] generates id attributes where they are missing. While this technique could be added to our approach, it is insufficient to handle all possible changes, e.g. when an element is moved within the internal DOM structure, or completely removed from the web site. In a related project, we are working on new robust implementation strategies that MultiMasher could build on in the future. In addition, if fully automatic solutions are not possible, the user could be informed and asked for suggestions. As an alternative, we are currently experimenting with a new selection method allowing users to select a region in the rendered web interface by spanning a rectangle with the mouse or finger. This region is then distributed and handled as a new window showing only the selected parts of the web site. However, as some elements may only be visible under certain viewing conditions, this method also needs to take into account that web sites may render differently on different devices, which requires further work.

6 Conclusion and Future Work

Based on a common scenario in which multi-device mashups would be desirable, we have presented our ongoing work on MultiMasher, a visual tool for rapidly developing multi-device mashups. MultiMasher provides an easy selection mechanism for distributing and mashing up UI elements, and is suited for an iterative design process as it implements first mechanisms for saving, loading, and merging multi-device mashups.

MultiMasher already provides much of the required functionality for quick and easy creation of mashups spanning multiple devices. However, at this stage, the resulting applications are not fully suited for a production environment. For example, as the main focus was on rapid prototyping of distributed mashups, we are currently relying on relatively simple mechanisms for the distribution of interfaces and the propagation of interactions. Moreover, privacy and security issues were out of scope. Some of the challenges could be addressed by introducing the notion of users in different roles and providing additional infrastructure. For example, a proxy that acts as the single client to the original web source could be used. However, rather than providing a generic and complex proxy-based solution that can handle all different kinds of scenarios, our goal for future versions of MultiMasher is that tailored proxy clients would be generated as part of the deployment of a multi-device mashup.

Another direction we can see for the future is improving the support for both experienced developers of distributed interfaces and non-technical end-users. This includes building new mechanisms into MultiMasher to support the evaluation of mashups in both static and dynamic multi-device environments.

For example, a user testing mode could be added that logs the interaction of the user with the mashup. As most user interactions are already tracked and sent to the server to keep the distributed interface synchronised, interactions could be recorded and later replayed for further analysis of the users' activities as well as for user performance evaluations in distributed, multi-device scenarios.

References

1. Paternò, F., Santoro, C., Spano, L.D.: Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. TOCHI 16(4), 19:1–19:30 (2009)
2. Ghiani, G., Paternò, F., Santoro, C.: On-Demand Cross-Device Interface Components Migration. In: Proc. MobileHCI (2010)
3. Melchior, J., Vanderdonckt, J., Roy, P.V.: A Model-Based Approach for Distributed User Interfaces. In: Proc. EICS (2011)
4. Paternò, F., Santoro, C.: A Logical Framework for Multi-Device User Interfaces. In: Proc. EICS (2012)
5. Daniel, F., Casati, F., Benatallah, B., Shan, M.-C.: Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. In: Laender, A.H.F., Castano, S., Dayal, U., Casati, F., de Oliveira, J.P.M. (eds.) ER 2009. LNCS, vol. 5829, pp. 428–443. Springer, Heidelberg (2009)
6. Daniel, F., Matera, M.: Turning Web Applications into Mashup Components: Issues, Models, and Solutions. In: Gaedke, M., Grossniklaus, M., Díaz, O. (eds.) ICWE 2009. LNCS, vol. 5648, pp. 45–60. Springer, Heidelberg (2009)
7. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target User Interfaces. IWC 15 (2003)
8. Melchior, J., Grolaux, D., Vanderdonckt, J., Roy, P.V.: A Toolkit for Peer-to-Peer Distributed User Interfaces: Concepts, Implementation, and Applications. In: Proc. EICS (2009)
9. Ghiani, G., Paternò, F., Santoro, C.: Push and Pull of Web User Interfaces in Multi-Device Environments. In: Proc. AVI (2012)
10. Huynh, D.F., Miller, R.C., Karger, D.R.: Potluck: Data Mash-up Tool for Casual Users. In: Aberer, K., et al. (eds.) ISWC/ASWC 2007. LNCS, vol. 4825, pp. 239–252. Springer, Heidelberg (2007)
11. Hartmann, B., Wu, L., Collins, K., Klemmer, S.R.: Programming by a Sample: Rapidly Creating Web Applications with d.mix. In: Proc. UIST (2007)
12. Maras, J., Štula, M., Carlson, J.: Reusing Web Application User-Interface Controls. In: Auer, S., Díaz, O., Papadopoulos, G.A. (eds.) ICWE 2011. LNCS, vol. 6757, pp. 228–242. Springer, Heidelberg (2011)
13. Cappiello, C., Matera, M., Picozzi, M., Sprega, G., Barbagallo, D., Francalanci, C.: DashMash: A Mashup Environment for End User Development. In: Auer, S., Díaz, O., Papadopoulos, G.A. (eds.) ICWE 2011. LNCS, vol. 6757, pp. 152–166. Springer, Heidelberg (2011)
14. Daniel, F., Soi, S., Tranquillini, S., Casati, F., Heng, C., Yan, L.: Distributed Orchestration of User Interfaces. Inf. Syst. 37(6), 539–556 (2012)
15. Heinrich, M., Grüneberger, F.J., Springer, T., Gaedke, M.: Reusable Awareness Widgets for Collaborative Web Applications - A Non-invasive Approach. In: Brambilla, M., Tokuda, T., Tolksdorf, R. (eds.) ICWE 2012. LNCS, vol. 7387, pp. 1–15. Springer, Heidelberg (2012)