SmartComposition: A Component-Based Approach for Creating Multi-screen Mashups

Michael Krug, Fabian Wiedemann, and Martin Gaedke

Technische Universität Chemnitz, Germany
{firstname.lastname}@informatik.tu-chemnitz.de

Abstract. The spread and usage of mobile devices, such as smartphones or tablets, increases continuously. While most of the applications developed for these devices can only be used on the device itself, mobile devices also offer a way to create a new kind of applications: multi-screen applications. These applications run distributedly on multiple screens, like a PC, tablet, smartphone or TV. The composition of all these screens creates a new user experience for single as well as for several users. While creating mashups is a common way for designing end user interfaces, they fail in supporting multiple screens. This paper presents a component-based approach for developing multi-screen mashups, named SmartComposition. The SmartComposition approach extends the OMELETTE reference architecture to deal with multiple screens. Furthermore, we enhance the OMDL for describing multi-screen mashups platform independently. We draw up several scenarios that illustrate the opportunities of multi-screen mashups. From these scenarios we derive requirements SmartComposition needs to comply with. A huge challenge we face is the synchronization between the screens. SmartComposition solves this through real-time communication via WebSockets or Peer-to-Peer communication. We present a first prototype and evaluate our approach by developing two different multi-screen mashups. Finally, next research steps are discussed and challenges for further research are defined.

Keywords: Mobile, distributed user interface, distributed displays, multi-screen applications, web applications, mashup, widgets.

1 Introduction

Internet-enabled devices are not anymore limited to computers and laptops. Devices, like smartphones, tablets or TVs, also offer users access to the Internet [8]. The number of mobile devices sold in 2013 exceeds the numbers of computers and laptops by factor three [10]. These new devices as well as the new capabilities of the Internet enable applications that can be used in several areas, such as entertainment, communication or productivity. While most of these applications can only be used on the device itself, mobile devices offer a way to create a new kind of applications: multi-screen applications. Multi-screen applications run distributedly on different devices, like a PC, smartphone, tablet or TV. The

S. Casteleyn, G. Rossi, and M. Winckler (Eds.): ICWE 2014, LNCS 8541, pp. 236–253, 2014.

composition of these multiple screens creates a new user experience. Multi-screen applications are suitable for single as well as several users.

A particular kind of multi-screen applications are multi-screen mashups. These multi-screen mashups are also a special type of user interface mashups (UI mashups), the importance of which has increased significantly within the last years [1]. A mashup consists of several widgets that offer a limited functionality. By combining and aggregating these widgets more complex tasks can be solved. While each widget can work by itself, it has to be extended to support the communication with other widgets inside the mashup. Examples for these UI mashups are platforms like iGoogle or Yahoo! Pipes. The compositional way UI mashups are built eases the development of new rich web applications. Current approaches regarding UI mashups focus on applications that run on a single screen. They offer the opportunity to easily create end user interfaces [3] on different devices but not across these devices. Although the EU FP7 project OMELETTE has focused on telco service mashups, it proposed a reference architecture for UI mashups in [6,18]. This architecture facilitates the deployment of UI mashups to desktop as well as mobile devices, but cannot distribute the UI across several devices. Another outcome of this EU project was the Open Markup Description Language (OMDL) [18] which supports the process of designing, evolving and deploying UI mashups, but is also limited to single-screen mashups. Thus, we extend the OMELETTE reference architecture and the OMDL to deal with multi-screen mashups.

Developing web applications is a methodical process our approach needs to deal with. Therefore, aspects of software and web engineering have to be considered. The WebComposition approach [4] describes a way to design and develop web applications based on reusable components. These components can be exchanged or reused within other web applications. The purpose of this paper is to propose an approach to support end users in creating multi-screen mashups. Therefore, we want to extend the UI mashup approach to support multiple screens. We present an architecture that supports and eases the development process of multi-screen mashups.

The rest of the paper is organized as follows: In Section 2 we describe three scenarios to illustrate the use cases we focus on. From these scenarios we derive challenges that a solution has to deal with in Section 3. The SmartComposition approach is proposed in Section 4. We describe our extension of the OMELETTE reference architecture and the OMDL. A first prototype which follows the SmartComposition approach is demonstrated in Section 5. In Section 6 we evaluate our approach. We discuss related work in Section 7. Finally, we provide a conclusion and give an outlook towards future work in Section 8.

2 Scenarios

In this section we present three scenarios that illustrate where multi-screen applications can create a benefit for users. Using these scenarios we want to show what users could expect from such applications and what challenges exist that need to be solved.

Scenario 1. The first scenario focuses on media enrichment using multiple screens. While consuming a documentary on television, the user, called Amy, wants to see additional content about the currently watched show. This additional content could be a related article on Wikipedia, images from online photo services, like Google or Flickr, or where to buy items that were shown in the video. For easing the reading or enabling interaction with the additional content, it will be displayed on Amys smartphone or tablet, while she is still watching the documentary on her television. Figure 1 illustrates this scenario. While watching a documentary about the United Nations (cf. 1) the location of the UN headquarters in New York is shown on a map on Amys smartphone (cf. 2) as soon as the video reaches the scene where it is presented. In the same way other kinds of information, which are directly related to the current scene, are provided. Amy can easily decide what types of additional content she wants to receive. For deleting this information two use cases are applicable. First, the information about the UN headquarters' location will be removed from Amy's smartphone, if the documentary does not talk anymore about it. Second, by clicking on the information Amy can prevent the automatic removing of the information.

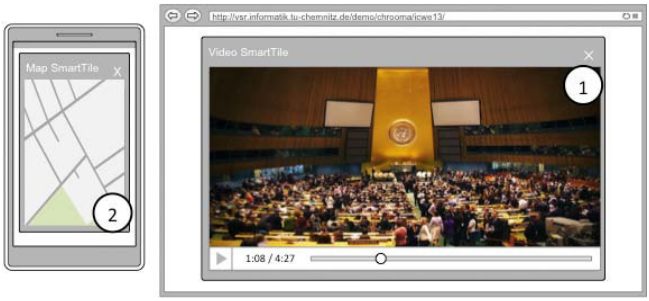

Fig. 1. Mockup illustrating media enrichment in Scenario 1

Scenario 2. The second scenario addresses the enhancement of a presentation by distributing additional information to multiple secondary screens. As an example, a professor is giving a talk in front of many students. While showing slides of his presentation via a projector, the students receive further information on their mobile devices. Thus, the students can read and see details about the current topic without messing up the slides with a lot of text. Furthermore, the students can use the additional information on their laptops as a script. The professor can keep his slides clean, while the students get a synced script related to the current lecture. While the professor is authenticated as a lecturer, he can broadcast slides and information. This functionality is denied to the students who do not have this privilege.

Scenario 3. Our third scenario aims at collaborative work on multiple screens. Therefore, we want to reuse a scenario described by Husmann et al. in [9]. In this scenario "two users, Alex and Bill, are planning a mountain biking trip at

home in their living room". While Alex is planning the mountain biking route on her tablet, Bill is searching for railway connections on his smartphone. Both are using the television in their living room to share the results of their tasks on a larger display. When Alex chooses a start point for their mountain biking trip, this start point will be pushed to Bill's smartphone, where it is automatically inserted in the input field for the destination of his railway connection query. Bill can choose his favorite railway connections to be displayed on the television. Alex' chosen mountain biking route will be displayed as a map on the television next to the railway connection information.

3 Analysis

We use the presented scenarios in Section 2 to derive objectives our approach needs to deal with for creating multi-screen mashups. First, we begin with some basic findings. In all scenarios there is always a common context shared between all participating screens. This context has to be maintained. We identified two possible screen constellations. In the first one a primary screen publishes information to all secondary screens (cf. Scenarios 1 and 2). The secondary screens are only consuming and displaying information. An interaction with the information is possible, but without reflection to the original publisher. In the second constellation all screens are in an equal role (cf. Scenario 3). All of the participants are publishing and consuming information. A more interactive and collaborative usage of the application is possible. This setup requires a more complicated synchronization mechanism. Furthermore, there are also two scenario types regarding the number of users: single-user (cf. Scenario 1) and multi-user (cf. Scenarios 2 and 3). Both will require a different application design.

Based on the scenarios in Section 2, we now state several objectives that our approach should fulfill.

Simplicity of Mashup Creation. We targeted several scenarios in Section 2 where multi-screen mashups can provide a suitable solution. Thus, our approach should enable end users to easily create these multi-screen mashups for their specific tasks. This includes the reuse of developed widgets as well as the easy configuration of a set of devices which interact as a multi-screen. Inspired by the WebComposition approach from Gaedke et al. in [4,5] we apprehend widgets as loosely-coupled components, which were assembled in a mashup to a new application. Furthermore, the end users should not fiddle with designing UI mashups for mobile use only. The process of creating a multi-screen mashup is independent of the targeted device which can be a mobile as well as a PC or laptop.

Support for Multi-screen Usage. The focus of our proposal is the development of multi-screen mashups. Therefore, our approach should provide functionality to develop applications that can be used with more than one screen or device. Thus, there is a need for unique identification and data synchronization. The types of the participating screens should be detected and used to deliver an

according user interface. While there can be a lot of devices which use the system, they should not be connected to any other available screen. This objective implies that devices can be grouped to so called workspaces and cannot communicate with devices in different workspaces. As we described in Scenarios 1 to 3 the support for multiple screen is the central point of our approach.

Support for Real-Time Communication and Synchronization. To propagate information without noticeable delay between server and client a component for real-time communication and synchronization should be available. We define the communication as real-time, when the latency of the transmission of a message from one widget to another one is 50 milliseconds or less [15]. That ensures no noticeable delay when distributing information across one screen as well as multiple screens. Furthermore, the components on one screen must be able to communicate among themselves.

Collaboration Support. The applications to develop should not only be used by a single user. We want to support the development of multi-screen multi-user mashups. Thus, our approach should provide the basis for collaborative work, such as user identification and data synchronization. This is highly related to the real-time synchronization requirement. Scenario 3 illustrates this objective best, while Scenario 2 also partially requires some support in collaboration.

Authentication. Nowadays it is important to provide a personalized user experience as well as social interaction. Thus it is required to offer the users a way to authenticate themselves. Using protocols like WebID even an authentication without requiring a password is possible. A user and role based identity management would provide a flexible way to handle most scenarios. The usage of multiple different devices creates a challenge to provide user-friendly login mechanisms for different use cases.

After we specified our objectives, we propose our own approach within the next section.

4 The SmartComposition Approach

To fulfill the requirements gathered from the analysis we propose a component-based approach to develop multi-screen mashups, called SmartComposition. We extend the OMELETTE reference architecture [18] for fitting the requirements in Section 3. In the following paragraphs we describe the SmartComposition architecture, which is depicted in Figure 2. We highlight our significant changes related to the OMELETTE reference architecture with underlined text. Our proposal consists of four parts: SmartComposition Runtime Environment, Multi-Screen Workspace, Information Store, SmartScreen. These parts interact like described in the following.

The SmartComposition Runtime Environment runs a multiple instances of a mashup. Thus, it handles multiple Multi-Screen Workspaces which are separated and secured against other existing Multi-Screen Workspaces. To enable the security between the Multi-Screen Workspaces we introduce the Session-Handler.

The Session-Handler knows all running workspaces and instantiates the Multi-Screen Workspaces. Each Multi-Screen Workspace contains a Workspace Manager.

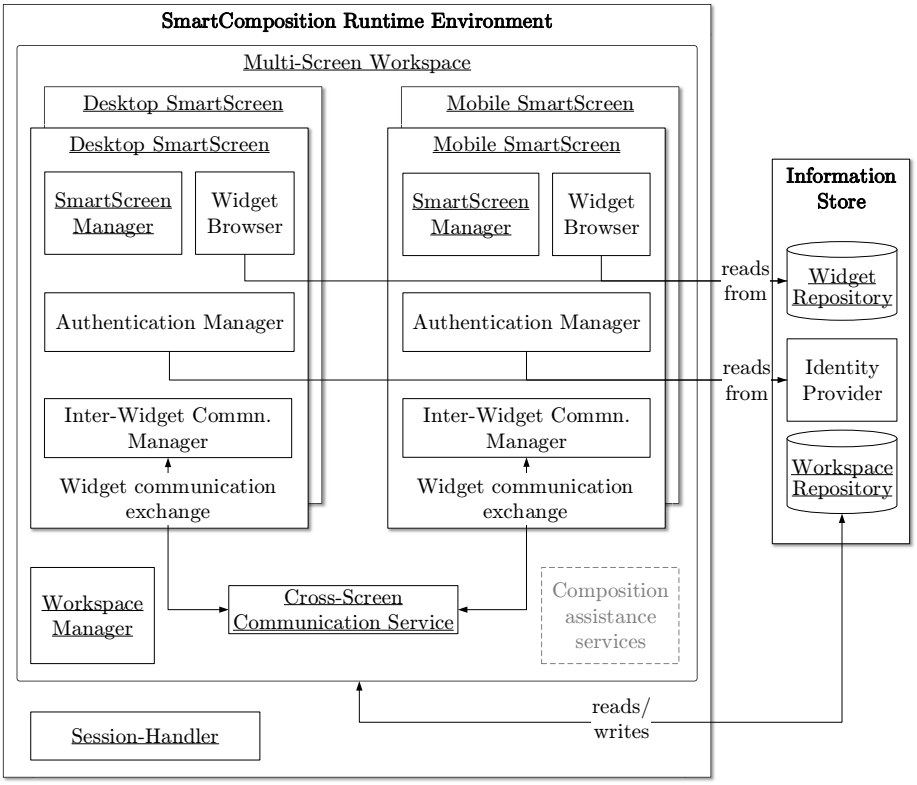

Fig. 2. Architecture of the SmartComposition approach

The Workspace Manager handles the workspace of a single user as well as several users who use the mashup in a collaborative way. Furthermore, the Workspace Manager is responsible for initialization of the SmartScreens and for the state management of the whole workspace. The illustrated component composition assistance services [16] in Figure 2 are not covered by our approach yet. Therefore, a workspace is defined as a set of one or more SmartScreens, which share the same information space and can communicate among themselves. Thus, the simplest form of a workspace consists of a single SmartScreen. A workspace is identified by a workspace ID and described by its workspace configuration. The configuration contains information about the SmartScreens which are connected, the widgets on each screen and additional properties. We use the OMDL as the basis for our configuration notation. Therefore, we extend the OMDL to support multi-screen aspects. How we did this and what changes we made is described later in this section.

The inter-widget communication is a highly important topic when developing mashups [1]. Our approach does not improve the inter-widget communication by itself but extends it for multi-screen mashups. On a single SmartScreen we use the publish-subscribe pattern for transmitting messages from one widget to one or several other widgets. For supporting inter-widget communication across multiple screens we extend the inter-widget communication component to publish the message to other connected SmartScreens (cf. Figure 3). That is, if a widget publishes a message on a certain topic, the inter-widget communication component publishes this message to all widgets on the same SmartScreen that are subscribed on the same topic. Furthermore, the message is transmitted to a Cross-Screen Communication Service where the message is sent to all connected SmartScreens within the same session. The assignment of SmartScreens and the workspace session they belong to is done by the SessionHandler. This ensures that a SmartScreen can only communicate with other SmartScreens of the same workspace. On each SmartScreen the inter-widget communication component publishes the message also to the subscribed widgets on their SmartScreen. The Cross-Screen Communication Service can work in two modes: broadcast mode and multi-layered publish-subscribe mode. In broadcast mode the Cross-Screen Communication Service sends every incoming message of one SmartScreen to all other SmartScreens of the same workspace. However, in the multi-layered publish-subscribe mode the inter-widget communication component of each SmartScreen subscribes to topics at the Cross-Screen Communication Service. The topics to subscribe depend on the topics that the widgets subscribe to on their SmartScreen. After publishing a message from a widget to the inter-widget communication component of its SmartScreen the message will be published to all subscribed widgets. Furthermore, the message will be published to the Cross-Screen Communication Service. There, the message will be published to inter-widget communication component of the SmartScreens, if they are subscribed to the topic. Each inter-widget communication component then publishes the message to the subscribed widgets.

As defined before, a workspace consists of SmartScreens. A SmartScreen is an abstract representation of a web browser window. It provides the runtime environment for all client-side components. Each SmartScreen has its own identifier. This identifier has to be at least unique in the workspace. This is important for the communication between the screens. The SmartScreen Manger within the SmartScreen initializes the widgets, handles the state management and provides access to device-specific properties. It detects the type and resolution of the device and uses this information to adapt the presentation. Thus, it is responsible for composition of the user interface.

The information store is the part which is dealing with all required and available persistent data. It consists of three components: the widget repository, the identity provider and the workspace repository.

Described in the OMELETTE reference architecture in [18] the widget repository is part of the SmartComposition Runtime Environment, but there are also meta data about the widgets available in the information store. Thus, we decided

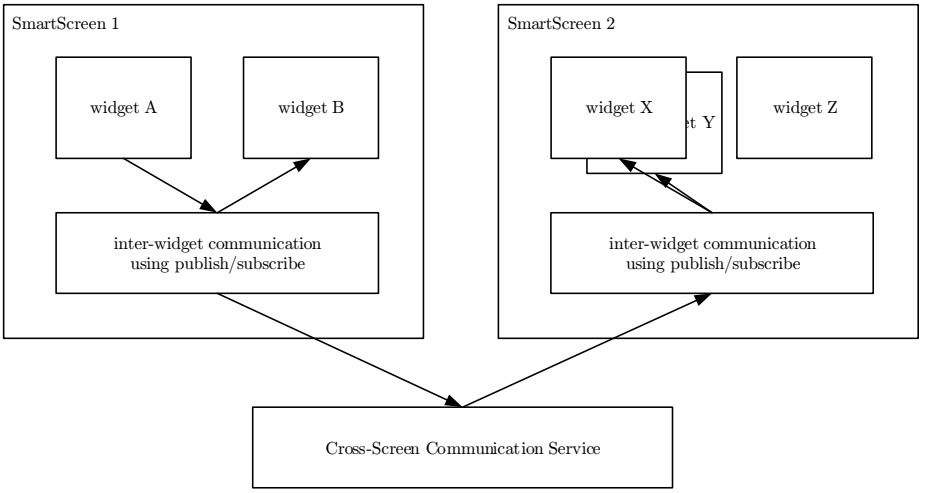

Fig. 3. Cross-screen message flow of inter-widget communication

to combine both parts and move the widget repository to the information store. There it is responsible for delivering the executables as well as providing meta data about the widgets. The end user can search for a specific widget based on the name or id of the widget. Furthermore, the end user can discover widgets based on their meta data. That is, a widget can be selected by the messages it can consume or the way it presents the proposed information, such as a map widget which shows given coordinates on a map excerpt. The widget repository is also responsible for delivering the executables to the SmartComposition Runtime Environment. In case of W3C widgets [19] the executables contains HTML-, CSS- and JavaScript-files which can be delivered to the SmartScreen without much effort.

The identity provider offers information about the current user or users. It contains endpoints for verifying the user's identity. While our approach enables using different identity management concepts, we propose the usage of WebID [17]. When using WebID the identity provider gives access to the users' WebID profiles. The supplied information about the authenticated user can also be used for accessing resources from the widget repository as well as the workspace repository [7]. Closely related to the identity provider is the Authentication Manager within the SmartScreens.

Therefore, the Authentication Manager requests the users WebID certificate. The WebID URI is extracted from the Subject Alternative Name of the WebID certificate. Then, the Authentication Manager requests the WebID profile from the Identity Provider and verifies that the public key of the users WebID certificate is equal to the public key in the corresponding WebID profile. This is the authentication part within the WebID flow described in [17]. Afterwards, based on the users WebID, authorization could be granted. Based on the authenticated user access to restricted resources can be permitted or denied, such as the users

workspace or some special widgets, which are limited to specific users. To ease the access to a users roles, the Authentication Manager offers an interface to request the users WebID profile for personal information, like name, birthdate and more. Furthermore, the users roles can be requested by other widgets or core components of the SmartComposition Runtime Environment. However, the users relationships can be queried without requesting the users WebID profile again.

The workspace repository stores and offers predefined or user-specific Multi-Screen Workspaces. Predefined workspaces will be used if there is no user-specific workspace existing. Thus, our approach enables the developer to define the default appearance of a single SmartScreen. Defining a default workspace with more than one SmartScreen is not useful, because the developer cannot predict with how many devices the user will use the application. However, it is possible to define multiple different default workspaces regarding to context specific information, such as the users role, device or geo-location. Within the default workspace a set of widgets and their position on the SmartScreen is defined.

When a user or several users are using the multi-screen web application, it could be necessary to save the current Multi-Screen Workspace. The users workspace will be saved by storing the OMDL description of the workspace in the workspace repository. While a Multi-Screen Workspace can contain multiple SmartScreens, it is at least necessary that one device wants to restore its SmartScreen as part of the workspace. Therefore, the device needs to save at least the workspace identifier within its client-side storage. The workspace repository offers an interface where the OMDL description corresponding to a given identifier is responded. From this OMDL description the device can extract its workspace configuration and can so restore the widgets on this SmartScreen. When another device of this workspace wants to restore its SmartScreen, it also sends the workspace identifier and retrieves the OMDL description of the workspace where it can extract its SmartScreen. On restoring the second or another SmartScreen the communication channel between these SmartScreen will be established by the Cross-Screen Communication Service.

4.1 Extension of OMDL

We use the OMDL for describing our Multi-Screen Workspaces and for storing them in the workspace repository. While the OMDL defines three levels for describing mashups [18], we just use the Physical Level for the SmartComposition approach. However, the defined Physical Level needs to be extended to support multiple screens, because the OMDL has no support for distributed mashups. The OMDL uses XML documents to describe mashups, their layout and the apps they contain. The root element of an OMDL document is the workspace. In the SmartComposition approach we apply the OMDL workspace to theMulti-Screen Workspace described above.

While the OMDL does not support multiple screens yet, we extend the vocabulary of the OMDL by adding several elements. The first new element is the SmartScreen which is added within the workspace element. The SmartScreen

element is used to define a SmartScreen within a Multi-Screen Workspace. Thus, there can be multiple SmartScreen elements. The SmartScreen element has an attribute id, which is unique and identifies the SmartScreen. Furthermore, the SmartScreen element has a child element named type, which defines the type of the device the SmartScreen runs on. For dealing with collaborative multi-user workspaces we define a user element within the SmartScreen element, which refers to the authenticated users WebID URI.

```
<workspace xmlns="http://omdl.org/">
 <identifier>http://example.org/workspacerepo/ws1</identifier>
 <title>Media</title>
 <date>2012-07-03T14:23+37:00</date>
 <SmartScreen id="phone1">
   <type>Smartphone</type>
   <user>http://example.org/people/alice.rdf#me</user>
   <grid><height>4</height><width>2</width></grid>
 </SmartScreen>
 <SmartScreen id="pc1">
   <type>PC</type>
   <user>http://example.org/people/bob.rdf#me</user>
   <grid><height>6</height><width>10</width></grid>
 </SmartScreen>
 <app id="http://example.org/workspacerepo/ws-of-alice/1">
   <SmartScreen>phone1</SmartScreen>
   <type>MAP</type>
   <link rel="source" href="http://example.org/repo/w1"
     type="application/widget"/>
   <position><x>0</x><y>0</y></position>
   <size><height>2</height><width>2</width></size>
 </app>
 <app id="http://example.org/workspacerepo/ws-of-alice/2">
   <SmartScreen>pc1</SmartScreen>
   <type>LIST</type>
   <link rel="source" href="http://example.org/repo/w2"
     type="application/widget"/>
   <position><x>0</x><y>0</y></position>
   <size><height>5</height><width>2</width></size>
 </app>
</workspace>
```

Listing 1.1. Example of a workspace description in OMDL

Each SmartScreen has an individual fine grained grid to arrange the widgets. To store the position of the widgets the SmartScreen element has a child element named grid. This element has two child elements height and width which represent the number of rows and columns the grid has. Another element we adapted is the app element. In OMDL this element describes a part of the mashup. In the

SmartComposition approach we define an app as a widget. While the OMDL has no multi-screen support yet, we also need to extend the app element to define on which SmartScreen the widget is located. Therefore, we introduce a SmartScreen element within the app element. This SmartScreen element contains the id of the containing SmartScreen. Furthermore, we assume that the type element of the app can be extended by several types which are required for a specific multi-screen web application. However, the alignment of apps in OMDL does not fit our approach, which focuses on a finer grained positioning. Therefore, we extended the position element by two new elements which are x and y. These elements represent the position of the widget regarding to a fine grained grid on the SmartScreen, where x means the distance to the left side and y means the distance to the top. Furthermore, we added a new element to the app element named size. This element has two child elements height and width, which mean the size of the widget regarding to fine grained grid on the SmartScreen. An example of a SmartComposition workspace described in OMDL can be seen in Listing 1.1.

5 Prototype

To validate the SmartComposition approach we have implemented a first prototype, which is based on another work we recently presented at the ICWE 2013 [12,14] and WWW 2014 [13]. The prototype demonstrates the Scenarios 1 and 2 we described in this paper (cf. Figure 4). All the client-side components we describe are implemented in JavaScript running as a web application in a web browser.

To achieve a multi-screen experience we implemented the following components. We created a SmartScreen class. This class works as described in our approach and is the host for the widgets. For our user interface components we implemented a basic widget class which provides the interface and basic functionality. This basic class is used to derive various new widget types using prototype-based inheritance. We have implemented widgets that can display a map, images from different web sources, excerpts from Wikipedia, text, translations or tweets. The widgets can be added and removed on runtime. The user can arrange them in a grid-based layout by using drag-and-drop.

To handle the information exchange between the widgets we implemented a component as part of the inter-widget communication. This component provides the publish-subscribe pattern and thus offers loosely coupled communication. Once a widget is added to the SmartScreen it can subscribe to one or more topics on which events are published.

The workspace configuration of the SmartScreens is currently stored on the client-side using the HTML5 feature LocalStorage. Using the proposed workspace identifier the users customized arrangement can be restored. A workspace repository is not yet implemented.

To make the prototype work on multiple screens we extended the inter-widget communication component with a synchronization mechanism. This mechanism uses WebSockets to propagate the events which were published on one

Fig. 4. Prototype of a multi-screen mashup described in Scenario 1. On the primary screen (top) a german news cast is played. Meta data from this video are published to all widgets within the workspace. Additional information are displayed on the different screens, such as Google Maps on the primary screen, Flickr images on the iPad (bottom right), and Twitter feed on the iPhone (bottom left).

SmartScreen to the other connected screens. This extension also handles the re-publishing of received events to the widgets. Thus, the whole communication is synchronized with all connected SmartScreens. This approach assures that each client behaves equally regardless of where the event was published. To make that work we also had to implement a server-side component. Once a new SmartScreen is added to a workspace the RTC components registers the screen at the Session-Handler using the workspace ID, the SmartScreens own identifier and device specific information. This data is supplied to the other participants.

To deal with the Cross-Screen Communication Service we implemented a WebSockets server that provides methods for propagating information to other connected clients. It handles the propagation of the events it receives to all SmartScreens that are combined in a workspace. Using the workspace identifier the related SmartScreens are selected. This component is implemented as a Node.JS application.

Demonstration. The prototype presented in this paper is available for testing at: `http://vsr.informatik.tu-chemnitz.de/demo/chrooma/icwe14/`

6 Evaluation

In this section we evaluate the SmartComposition approach. Based on the analysis in Section 3, we examine how the approach assists in developing multi-screen mashups. We outline our expectations and explain actual findings.

A big challenge our approach has to deal with is the support of multiple screens. Therefore, we introduced the concept of workspaces where several SmartScreens create the interface of the mashup. The introduced parts of our architecture, the Cross-Screen Communication Service and the Session-Handler, enable the inter-widget communication across multiple screen (cf. Figure 2). This two parts also ensure that only screens within the same workspace can communicate with each other. Thus, the workspaces are isolated from each other. We extended the OMDL to support multiple screens within a workspace and used it to store and restore the configuration of a workspace.

Real-time communication for the SmartScreens has to be enabled for synchronization and communication of multiple SmartScreens within a workspace. While web systems often use long-polling for nearly real-time communication, we considered that new technologies, such as WebRTC or WebSockets, which are introduced with HTML5, might better correspond to our requirements. Therefore, we designed our RTC component using WebRTC and WebSockets and support long-polling as a fallback mechanism for browsers, which do not support these new technologies. We evaluate the performance of the RTC component in our prototype by measuring the duration of transporting an information from one widget to another. A special widget was developed, which publishes a timestamp to a second widget. The second one re-publishes the timestamp to its source. After receiving the timestamp on the first widget the difference between the current timestamp and the received one is calculated and then halved. We examined different constellations. The first one was where both widgets were

Table 1. Transport duration of a message from one widget to another in milliseconds

	Average	**Standard Deviation**
Same SmartScreen	5,121	0,654
Same PC	4,100	0,921
PC - iPad	38,735	26,842
PC - PC (WiFi)	26,668	19,036
PC - PC (Wired)	3,494	1,254

on the same SmartScreen. Another one was where the widgets were on different SmartScreens which run on the same PC. The third constellation was one SmartScreen on a PC and the other on a WiFi-connected iPad. The fourth one was two SmartScreens on two PCs which are connected via WiFi. Two PCs are connected via wire in the last constellation. The statistical values of our evaluation can be seen in Table 1. As we proposed in Section 3 the real-time communication is satisfied as the transport of a message from one widget to another is less than 50 ms. Our findings show that the largest latency occurs at the communication between a widget on the PC and a widget on the iPad with 38,735 ms. We can argue the difference in the average communication between the same SmartScreen and two SmartScreens on the same PC with the multi-threading architecture of the PC. That is, with two SmartScreens in two browser windows enables more parallelism in execution, because each browser window can be run in a different thread. The high standard deviation in the communication via WiFi (PC-iPad and PC-PC) can be ascribed to some side effect caused by the WiFi. To ensure that the higher latency when using WiFi does not is caused by constant term we give the relation of the IP-latency measured by ping in Table 2.

Table 2. Ping latency between two PCs in milliseconds

	Average	**Standard Deviation**
PC - PC (WiFi)	3,800	1,279
PC - PC (Wired)	1,025	0,798

We evaluate the simplicity of creation by developing two mashups for Scenario 1 and Scenario 2. In Scenario 1 we developed some new widgets for displaying a video and processing the meta data, such as the given subtitles. Other widgets could be reused in both scenarios, like a Wikipedia-widget which is displaying an article from Wikipedia or a GoogleMaps-widget which shows a map excerpt. Since the separation of the SmartScreens into their workspaces is an essential part of our approach, the end user has not to deal with the configuration of the workspaces. A mashup created based on our approach runs on PCs, laptops, smartphones, tablets as well as on web-enabled TVs.

For supporting authentication we propose a two tier approach The Authentication Manager within the SmartComposition Runtime Environment deals with the process of authentication and offering interfaces for accessing roles and permissions. The Identity Provider in the Information Store gives access to the users' profile and personal data, such as name, birth date or e-mail address. The usage of WebID offers the opportunity to let users easily login with their different devices. They can use a WebID certificate on each device which refers to the same WebID profile of the user.

Performing collaborative tasks is facilitated by multi-user workspaces and real-time communication. For collaborative work on a multi-screen web application we extended the OMDL to define a specific user for each SmartScreen. Thus, a device, which is authenticated by the users WebID, receives the corresponding workspace and can load the SmartScreen associated to the device. This enables several users to use the same workspace. Utilizing the real-time communication functionality of the RTC component synchronization of different users SmartScreens can be achieved.

7 Related Work

Our approach is closely related to work in three research domains: distributing and migrating Web UIs, data mashups, and classic user interface mashups (UI mashups). The DireWolf approach proposed in [11] enables the distribution and migration of Web widgets between multiple devices. While our approach focuses on developing multi-screen mashups from scratch, the DireWolf approach extends the functionality of single-device mashups to run across multiple devices. The framework is involved in every layer of a widget-based Web Application: the widget itself, the client browser, the backend service and the data storage. DireWolf manages communication between widgets on one device as well as between widgets on multiple devices. It also extends the functionality of common widget spaces with a shared application state. While our approach uses HTML5 features, such as WebSockets, for communication the DireWolf approach uses the XMPP protocol and its publish-subscribe extensions. That is, the DireWolf approach uses an additional layer within the communication architecture, which increases the effort for maintaining and exchanging parts of this communication layer. Furthermore, the usage of XMPP can increase the incompatibility when using mobile devices inside restricted GSM networks.

MultiMasher is a visual tool to create multi-device mashups from existing web content [9]. It aims at designing mashups without the need for modeling or programming. The MultiMasher runs in the browser, connects to the MultiMasher server and provides the user with a toolbar. After the user has connected his devices he can load any web site, which he can then mashup. UI elements can be selected visually and distributed (move or copy) to other connected devices. The mashups can be saved, loaded and reused as basis for other mashups. MultiMasher uses event forwarding to propagate changes to the connected clients and only displays the previously selected UI elements on each device. MultiMasher focuses on the creation of mashups and not on the creation of new applications.

The EU FP7 Project OMELETTE is closely related to our work, because it proposes a reference architecture for designing platforms for UI mashups [18]. They focus on end users with less or no programming skills to create their own mashups for fulfilling a certain goal. The outcome of this project was a prototype based on Apache Rave and Apache Wooky [2]. After a user log in, she can restore recent workspaces or create a new one by adding widgets to her workspace. The focus of the OMELETTE project was on the automatic composition of these mashups. Therefore, they invented some great strategies for suggesting widgets to add or for establishing the inter-widget communication between two widgets based on the user's behavior. While the OMELETTE reference architecture is a well-proven approach, it lacks in supporting UI mashups which run distributed across multiple screens. Thus, we extend the proposed reference architecture to support workspaces which include multiple screens. We also showed that concepts of inter-widget communication designed for a workspace on one screen works also for a workspace across multiple screens.

Mashup tools, like Yahoo! Pipes, offer end users developing data mashups. End user in this context means people with no or less development skills. While our approach focuses on user interface mashups, Yahoo! pipes is focusing on mixing popular data feeds to create data mashups via a visual editor. Therefore, it lets users transform, aggregate, transform and filter one or more data sources, such as RSS/Atom feeds or XML sources, and output this as a RSS feed. Thus, it enables end users to develop a kind of business logic for processing data [20]. However, Yahoo! Pipes does not offer a multi-screen environment where the created mashups can be executed.

8 Lessons Learned and Outlook

The SmartComposition approach proposed in this paper enables end users to easily create multi-screen mashups. We extended the OMELETTE reference architecture to deal with mashups across multiple screens. The Cross-Screen Communication Service extends classic inter-widget communication to work on a trans-screen level, while separating different workspaces from each other via the Session-Handler. Furthermore, our approach handles the challenge of real-time communication. The component-based architecture enables adding required and removing obsolete components. Thus, SmartComposition can be adjusted for different multi-screen web application scenarios. Both, the workspace repository as well as the widget repository offer a generalized access to widgets and workspaces.

Our future work will focus on integrating commonly used widget formats, such as W3C-widgets or Opera widgets. We also plan to evolve the cross-screen inter-widget communication by considering constraints given by several roles. That is, in a multi-user scenario, e.g., Scenario 3, one user is permitted to update a specific widget on a shared screen, while another user just has read-only access to the presented information. Another aspect we want to focus on in future work is the authorization of widgets to use the users' external services, such as Facebook or Google Drive.

As described in Section 4 the composition assistance services, such as automatic composer and workspace pattern recommender, are not yet a part of our approach for multi-screen mashups. We plan to adapt the approaches proposed by Roy Chowdhury in [16] for multi-screen mashups. Different use cases in this field of research are possible: The user gets a recommendation to add an additional screen, like a smartphone or a laptop, when she is using the mashup to accomplish her desired goal in a better way. When a user starts a new workspace, the widgets will be deployed to all available screens. This can be done by learning from the behavior of the users or by designing a default workspace.

Acknowledgment. This work was supported by the Sächsische Aufbaubank within the European Social Fund in the Free State of Saxony, Germany (Project Crossmediale Mehrwertdienste für die digitale Mediendistribution).

References

1. Chudnovskyy, O., Fischer, C., Gaedke, M., Pietschmann, S.: Inter-widget communication by demonstration in user interface mashups. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 502–505. Springer, Heidelberg (2013)
2. Chudnovskyy, O., Nestler, T., Gaedke, M., Daniel, F., Fernández-Villamor, J.I., Chepegin, V., Fornas, J.A., Wilson, S., Kögler, C., Chang, H.: End-user-oriented telco mashups: the omelette approach. In: Proceedings of the 21st International Conference Companion on World Wide Web, pp. 235–238. ACM (2012)
3. Chudnovskyy, O., Pietschmann, S., Niederhausen, M., Chepegin, V., Griffiths, D., Gaedke, M.: Awareness and control for inter-widget communication: Challenges and solutions. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 114–122. Springer, Heidelberg (2013)
4. Gaedke, M., Rehse, J.: Supporting compositional reuse in component-based web engineering. In: Proceedings of the 2000 ACM Symposium on Applied Computing, vol. 2, pp. 927–933. ACM (2000)
5. Gaedke, M., Turowski, K.: Specification of components based on the web-composition component model. In: Managing Information Technology in a Global Economy, p. 411 (2001)
6. Gebhardt, H., Gaedke, M., Daniel, F., Soi, S., Casati, F., Iglesias, C., Wilson, S.: From mashups to telco mashups: A survey. IEEE Internet Computing 16(3) (2012)
7. Hollenbach, J., Presbrey, J., Berners-Lee, T.: Using rdf metadata to enable access control on the social semantic web. In: Proceedings of the Workshop on Collaborative Construction, Management and Linking of Structured Knowledge (CK 2009), vol. 514 (2009)
8. Horizont: Report TV-Marketing, Ausgabe 17, p. 40 (April 2012), `http://www.horizont.net/report`
9. Husmann, M., Nebeling, M., Norrie, M.C.: MultiMasher: A visual tool for multi-device mashups. In: Sheng, Q.Z., Kjeldskov, J. (eds.) ICWE Workshops 2013. LNCS, vol. 8295, pp. 27–38. Springer, Heidelberg (2013)
10. IDC Coroporate USA: Tablet Shipments Forecast to Top Total PC Shipments in the Fourth Quarter of 2013 and Annually by 2015, According to IDC (2013), `http://www.idc.com/getdoc.jsp?containerId=prUS24314413`

11. Kovachev, D., Renzel, D., Nicolaescu, P., Klamma, R.: DireWolf - distributing and migrating user interfaces for widget-based web applications. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 99–113. Springer, Heidelberg (2013)
12. Krug, M., Wiedemann, F., Gaedke, M.: Media enrichment on distributed displays by selective information presentation: A first prototype. In: Sheng, Q.Z., Kjeldskov, J. (eds.) ICWE Workshops 2013. LNCS, vol. 8295, pp. 51–53. Springer, Heidelberg (2013)
13. Krug, M., Wiedemann, F., Gaedke, M.: Enhancing Media Enrichment by Semantic Extraction. In: Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion, pp. 111–114. International World Wide Web Conferences Steering Committee (2014)
14. Oehme, P., Krug, M., Wiedemann, F., Gaedke, M.: The chrooma+ approach to enrich video content using html5. In: Proceedings of the 22nd International Conference on World Wide Web Companion, pp. 479–480. International World Wide Web Conferences Steering Committee (2013)
15. Pantel, L., Wolf, L.C.: On the impact of delay on real-time multiplayer games. In: Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video, pp. 23–29. ACM (2002)
16. Roy Chowdhury, S., Chudnovskyy, O., Niederhausen, M., Pietschmann, S., Sharples, P., Daniel, F., Gaedke, M.: Complementary assistance mechanisms for end user mashup composition. In: Proceedings of the 22nd International Conference on World Wide Web Companion, pp. 269–272. International World Wide Web Conferences Steering Committee (2013)
17. Sporny, M., Inkster, T., Story, H.: WebID 1.0: Web Identication and Discovery (2011), http://www.w3.org/2005/Incubator/webid/spec/
18. University of Trento: D2.2 - Initial Specification of Mashup Description Language and Telco Mashup Architecture. Tech. rep., University of Trento (2011)
19. W3C: Packaged Web Apps (Widgets) - Packaging and XML Configuration, 2nd edn. (2012), http://www.w3.org/TR/2012/REC-widgets-20121127/
20. Yu, J., Benatallah, B., Casati, F., Daniel, F.: Understanding mashup development. IEEE Internet Computing 12(5), 44–52 (2008)