Journal of Web Engineering, Vol. 13, No. 3&4 (2014) 203–222

DIREWOLF: A FRAMEWORK FOR WIDGET-BASED DISTRIBUTED USER INTERFACES

DEJAN KOVACHEV, DOMINIK RENZEL, PETRU NICOLAESCU,
ISTVÁN KOREN and RALF KLAMMA
Advanced Community Information Systems (ACIS) Group, RWTH Aachen University
Ahornstr. 55, Aachen, 52056, Germany
{*kovachev|renzel|nicolaescu|koren|klamma*}*@dbis.rwth-aachen.de*

Web applications have overcome traditional desktop applications especially in collaborative settings. However, the bulk of Web applications still follow the "single user on a single device" computing model. Therefore, we created the DireWolf framework for rich Web applications with distributed user interfaces (DUIs) over a federation of heterogeneous commodity devices supporting modern Web browsers such as laptops, smart phones and tablet computers. The DUIs are based on widget technology coupled with cross-platform inter-widget communication (IWC) and seamless session mobility. Inter-widget communication technologies connect the widgets and enable real-time collaborative applications as well as runtime migration in our framework. We show that the DireWolf framework facilitates the use case of DUI-enabled semantic video annotation. For a single user it provides more flexible control over different parts of an application by enabling the simultaneous use of smart phones, tablets and computers. We conducted a technical evaluation and two user studies to validate the DireWolf approach. The work presented opens the way for creating distributed Web applications which can access device specific functionalities such as multi-touch, text input, etc. in a federated and usable manner. In this paper, we also sketch our ongoing work to integrate the WebRTC API into DireWolf, where we see opportunities for potential adoption of DUI Web applications by the majority of Web users.

Keywords: distributed user interface; Web widget; inter-widget communication; XMPP; WebRTC

1 Introduction

People increasingly interact with a collection of heterogeneous computing devices attached to their daily lives. However, most Web applications fail to combine devices' features into a cohesive symbiotic way to convey a single user task in a collaborative fashion. One of the reasons behind this failure is the lack of tools and methodologies required to develop applications spreading user interfaces across multiple devices available to a particular user or group of users. Personal computing is no longer confined to a single device. PCs together with commodity smartphones, tablets, eBook readers, gaming consoles and interactive TVs can be federated over the Internet to create collaborative multi-device interactive systems which can benefit from the diverse device capabilities. An individual can interact in different ways with such symbiotic computing environments consisting of personal devices.

As a consequence, monolithic single-device *user interfaces (UI)* devolve to *Distributed User Interfaces (DUI)*. DUIs separate, migrate and merge seamlessly between devices. Additionally,

they can adapt to different platforms [1] and account for changes in device availability to achieve a continuous application experience [2].

Developing distributed user interfaces is challenging [3]. From the user perspective, two challenges are salient. First, users should be supported to adapt the distribution to their needs. Second, users should experience seamless UI migration. Migrated UI components preserve state and remain consistent with the whole application context. Concerning the use of multiple devices, current Web applications can be well rendered on different platforms. However, most of them ignore the possibility of using multiple personal computing devices. Cooperation between such devices related to distributed interfaces is scarce and mostly limited to device-specific static interface separation.

Fig. 1. An example of distribution of user interface components (widgets) to diverse (mobile) computing devices

To address these challenges, we present DireWolf, a framework for distributed Web applications based on widgets. Similar to the general requirements of such frameworks given in [4], with DireWolf we also strive to increase the number of input modalities and extend the display area of a personal computing environment by using a federation of devices (desktops, laptops, smartphones, tablets). However, we chose to limit the support to Web based applications which are installation-free and cross-platform. We have chosen to work with Web widgets because they represent interface components with limited, but clear-cut functionality, dedicated to smaller tasks. Widgets can be shared, reused, mashed up and personalized

between applications. By splitting the interface into separate widgets and enabling them to exchange information, customizable Web applications can be developed. Whereas previous work [5, 6] on widget applications and mashups considers single-end devices only, we examine the concept of widget-based Web applications combined with device awareness, session mobility and cross-device cooperation.

To illustrate the concept, we shortly describe a semantic video annotation application (cf. Figure 1). This application was transformed from a typical Web application into a widget-based one, thus validating the feasibility of our approach. A semantic video annotation application is an ideal candidate for extended UI interactions: users watch videos, annotate them at certain time points or for specific time intervals and navigate through a video using the annotations. Various types of available semantic annotations (agent, time, concept, object type) can be added using text input and interacting with a video player. Place annotations can be pinpointed on a map. However, e.g. full screen mode of the video player hides all other UI controls on one device. In an annotation scenario, distributing the UI enhances user experience. Users can play the video in full screen on one device and can use additional devices to annotate it or to browse through the video. Moreover, they can use device-specific features for each of the UI elements, e.g. multi-touch on a smartphone for interacting with a digital map. Preserving UI state across devices is also required for such a scenario, e.g. resume at current position instead of restart after migration of a video player, continue annotating, etc. Our paper brings forward the following contributions:

- a framework for easy browser-based distribution of Web widgets between multiple devices,
- support for extended multi-modal real-time interactions on a federation of personal computing devices,
- provisioning of continuous state-preserving widget migration,
- a performance evaluation of widget migration, and usability studies of widget-based DUI Web applications.

DireWolf helps managing a set of devices and handles communication and control of distributed parts of the Web application. The conceptual and implementation details of the DireWolf framework, together with the possibility of integration into existing widget platforms are detailed in the next sections.

The rest of the paper is structured as follows. In Section 2, we present relevant literature related to our approach. In Section 3 we introduce current widget-based Web applications as a starting point for our DUI framework. Sections 4 presents the DireWolf framework in detail with a focus on the framework concept and continuous widget migration (i.e. preserving the widget state before and after the migration). Section 5 provides implementation details. Results from the technical evaluation and user studies are discussed in Section 6. Finally, Section 8 concludes this work and provides an outlook to future research.

2 Related Work

Our DUI approach is related to work in two research domains, namely mechanisms for distributing and migrating Web UI and frameworks for using multiple personal computing devices to perform a single user task.

Distributing Web UIs means ungrouping Web document elements and presenting them separately without compromising application functionality. The granularity of UI splitting can range from arbitrary partitions to pre-defined UI blocks. Ghiani et al. [7] provide a mechanism to select a part of a Web page which can be migrated and shown on a mobile device. However, this approach is only feasible for the adaptation of Web pages and does not support presentation of different UI components on multiple devices at the same time. Model-based approaches [2, 8, 9] define different abstract UI configurations at design time and generate concrete UI presentations at runtime. These works demonstrate dynamic distribution of Web interfaces among heterogeneous platforms. Learning to use the schema for an application induces additional development effort. Moreover, if a new application joins the system, new UI schema files must be written, and the root UI schema must be modified. In contrast, we consider Web applications composed of widgets using open Web standards.

DUI systems can be further discriminated according to the architectural approach. Centralized DUI systems manage UI distributions using one server which is in charge of all UI distribution activities, whereas client devices only display distributed UIs or meta-UIs. A typical centralized DUI system is proposed by Vanderhulst et al. [2]. Proxywork [10] is another example of a centralized approach which uses a proxy between the user devices and the actual Web application. The proxy modifies the application and places UI components on different devices. Thus, this approach avoids schema dependability. However, it requires an intermediate server to broker between end clients and the Web services. This works well with simple static Web pages, but may have difficulties with highly dynamic Web applications. In contrast, DireWolf also uses a server, but only for messaging between the UI components.

Alternatively, the distribution of UIs can be done in a *peer-to-peer (P2P)* manner [11]. All peers in the scope of the application are equal and they manage UIs without any global knowledge. However, considering the UI components created all over the network, the stability of the application cannot be guaranteed when clients with master proxies are disconnected. Moreover, all devices must install framework binaries before they can create DUI components or import remote components directly from other devices. Nevertheless, the emerging *Web Real-Time Communication (WebRTC)* [12, 13] API standard for direct browser-to-browser communication in modern browsers will help mitigate this issue.

Dynamic DUIs should support runtime component migration. Necessary steps for a successful migration are presented in the Roam project [14]. Roam preserves the application execution state information such as heap, stack, network sockets, etc. at the start of the migration and restores them after migration. For continuous Web browsing, Alapetite et al. [15] migrate Web sessions across mobile devices using 2D-barcodes captured by cameras. A dedicated State Mapper is also developed in [16] for state recovery during UI migration between mobile phones and digital TVs. Inspired by these approaches, our framework realizes complete continuous migration tailored to Web widgets.

Multi-device collaboration means that multiple devices can join the same application scope and that these devices can complete tasks together. Early approaches have focused on supporting desktop applications with devices such as PDAs and handheld computers over wired or wireless connections. Pebbles [17] extends computing and I/O functionalities by involving heterogeneous devices. The extended UIs are native applications specially tailored for each computing platform and each functionality. Thus, multi-device UI are tightly coupled with

the computing hardware. Melchior et al. [11] present a P2P framework that helps deploy distributed graphical user interfaces. All devices must install the framework before they can create components or import remote components directly from other devices. Many projects consider one-to-one mappings between users and devices, which is more applicable for collaborative scenarios. MarcoFlow [18, 6] uses modular UI to represent the relevant controls and information to the user, but it focuses on the orchestration of business processes involving multiple users with different data views. Pierce and Nichols [19] use the idea of ownership to address personal computing devices and to enable seamless user experience over multiple devices. Their prototype simplifies the development of applications that are aware of a user's devices but it does not support UI migration. The DireWolf framework supports any device with an available modern Web browser. There is no need for pre-installed components or configurations. In the following, we first introduce Widget-based Web applications to clarify the context in which DireWolf was developed.

3 Widget-based Web Applications

Important prerequisites for distributing individual elements of complete Web applications are a clear separation into conceptual and functional units, a context for managing separation, and cross-device communication between these units. In this section we briefly introduce *widget-based Web applications* and discuss why they fulfill the above prerequisites and thus served as foundation for the DireWolf framework.

The basic building block is a *widget*. Conceptually, a widget is a self-contained mini-application with limited, however clean-cut functionality. Widgets are usually designed to accomplish small stand-alone tasks, which may recur in multiple different applications. Furthermore, widgets are usually designed with limited display size, such that multiple widgets fit on one desktop browser screen or single widgets fit on limited-size mobile device screens. By design, widgets are reusable for multiple purposes in different applications. As such, widgets strongly resemble mobile applications. Technically speaking, existing widget standard specifications define widgets as packaged Web applications including means of configuration and access to dedicated widget application programming interfaces. Principally, any existing Web application can be "widgetized". However, the form factor of limited display size often requires an adapted design. In practice, widgets usually serve as minimal frontends to more complex Web services. For our work, widgets perfectly serve as the functional units to be migrated across devices.

Complex applications can be achieved by orchestrating multiple widgets in a dashboard fashion in *widget containers*. Research towards the effective integration of widgets to complete collaborative Web applications resulted in additional layers on top of widget containers that make use of the DireWolf framework, i.e. *widget spaces* [5] and *inter-widget communication*.

First, combinations of multiple widgets require a working context and technical support to manage such contexts. In our work, we employ the concept of a *widget space* as working context. A widget space is a collaboration context, in which multiple users collaboratively manage and operate sets of widgets and additional resources to create custom applications for different purposes. For this work, we extended widget spaces by the additional notion of multiple devices per user. Second, the integration of multiple widgets to complete applications requires an interoperable communication mechanism between widgets, referred to as *Inter-*

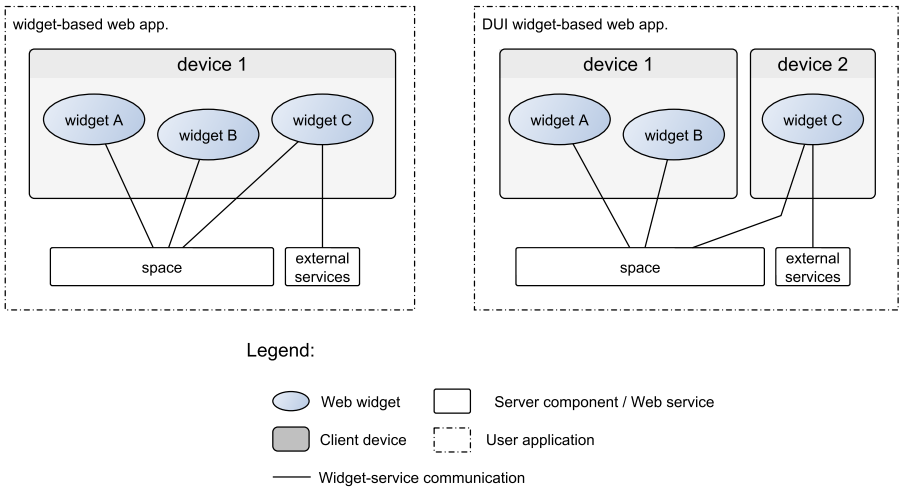


Fig. 2. Widget-based Web applications: (left) traditional non-distributed approach and (right) DUI approach

widget Communication. With such a usually publish-subscribe-based mechanism, messages can be broadcast from any widget and possibly dispatched by other widgets, thus allowing the combination of multiple widgets to complete integrated applications. Our approach thereby clearly falls into the category of choreographed UI mashups [20]. Most existing approaches only support local IWC, i.e. communication between widgets within one single browser instance. An additional feature of our complete IWC approach includes remote communication between widgets across different browser instances and users [21]. For this work, we use both forms of IWC as carrier for message exchange between different parts of our DUI framework within and across devices.

The left part of Figure 2 depicts the initial setting from which this work departed. In the following section we elaborate on the extensions contributed by our DUI framework in detail, thus leading to the situation in right part of Figure 2.

4 DireWolf Framework

Based on the state-of-the-art in widget-based Web applications discussed in the previous section, we now introduce the DireWolf framework. First, we discuss the particular requirements for such a framework, which are not yet covered by existing widget-based Web application frameworks.

4.1 Requirements Analysis

As a first step, we performed a requirements analysis with the goal of improving deficiencies found in existing work on DUIs (cf. Section 2), thereby taking into account the current state-of-the-art in widget-based Web applications (cf. Section 3). Figure 3 provides a high-level overview of the main identified requirements to managing distribution and migration in a DUI framework, grouped into four interrelated categories: *device information*, *device ownership*, *application state* and *widget handling*.

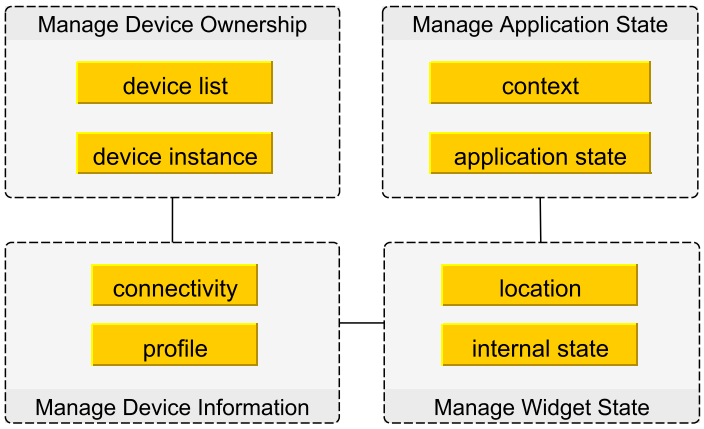


Fig. 3. Main requirements to managing distribution and migration in a dynamic widget-based DUI framework

A DUI framework must enable the management of general and context-specific device information. General information includes information on device *connectivity* and *profile*. A device profile captures information on device type (e.g. smartphone, tablet, laptop) and capabilities (e.g. operating system, display size, in/output modalities, browser type) required for device recognition and adaptation purposes. Device connectivity describes the current availability of the device for collaboration and should be updated in real time. Context-specific information includes *device location*, i.e. in which context the device is currently active, and *displayed widgets*, i.e. which widgets are displayed on the device in the current context.

Furthermore, a DUI framework must dynamically capture and manage *device ownership*. With the ever dropping prices of mobile devices, a person's device portfolio is likely to change often. Each user should thus be enabled to dynamically manage a personal *device list*. Thereby, each device instance describes a virtual device which can be bound to a real device. The introduction of virtual devices provides additional flexibility, i.e. multiple configurations for a single device and switching between real devices.

Obviously, a DUI framework must support distribution and migration of widgets across devices within a given context. In its simplest form, migration is a synchronized procedure controlled by the framework, where a widget is first removed from a source device and then created on a target device. However, constellations of widget distributions must be persistent. Thus, a DUI framework must be enabled to manage, store and synchronize application state within a given working context. For simple migration, application state must include information on the context and on widget locations, i.e. which widgets are currently residing on which device for which person. However, simple migration does not guarantee a seamless working experience. Although general widget configuration parameters are persistently managed by current standard widget engines, a widget will lose its internal state during the migration procedure. For some widgets this is not an issue (e.g. a clock widget), for some it is. Thus, a DUI framework must support the management, storage and synchronization of internal widget state. With such measures, a DUI framework is enabled to support continuous migration, i.e. a widget stores a snapshot of its internal state before removal from a source

device and restores internal state after its creation on the target device.

4.2 Framework Design

The DUI framework is involved in every layer of the widget-based Web application. Components should be created for widgets, client browsers, backend services as well as the data storage. Framework client components are included in the widget application document rendered in the Web browser. They manage communication and synchronization between widgets on one device, but also between widgets on other devices. The framework server components extend the functionality of common widget spaces with services for data persistence, user device profiles, and shared application state.

The DUI framework provides management services for device profiles and widgets when the user owns multiple devices. The inner workings of a widget are out of concern of the DUI framework. A requirement is that a mobile device needs to host some modern Web browser such as those found on most commodity smartphones and tablets. The use cases focus on creating, getter/setter and operating on resources (widgets).

Figure 4 depicts the key architecture features of the DireWolf framework. As mentioned in Sec. 3, the DUI framework requires a real-time communication mechanism to "glue" all distributed UI components into one cohesive application. The *Message Router* server component provides bi-directional asynchronous message exchange between the client components and the server.

DUI Client is a widget helper component to be included as a JavaScript library in the widget namespace. DUI Client usage in widgets is optional (e.g. legacy widgets). These widgets can still be distributed and migrated. However, the DUI Client enhances DUI-related features for the widget and provides an API to interpret and create framework messages and events. DUI Client has additional methods to store widget state as part of application state at the server-side service component. It sends requests, and server components send back responses as well as broadcast notifications to all other Web clients if necessary.

DUI Manager is the central DUI component on the client browser. All features resp. functionalities are directly or indirectly related to it. DUI Manager connects to other components of the framework in three ways: request-response communication, local and remote IWC. For example, DUI Manager uses requests-response communication to retrieve user profile and space information from server-side services. Local IWC is used for communicating with widgets running in the same browser context. Remote IWC provides the message-exchange mechanism for widgets and DUI Managers located at different devices.

At start, the DUI manager fetches the user profile which contains the device list and the device profiles. The connectivity of a user's devices is monitored constantly after the DUI manager is activated. The user can choose one virtual device per real device. If a device is not listed, the framework attempts to recognize it by using cookies, HTTP User-Agent headers and user input.

DUI Responder is the server-side central DUI component. All DUI relevant requests are redirected to this component. The main tasks of DUI Responder are to maintain DUI-relevant data and keep all DUI managers on client browsers synchronized.

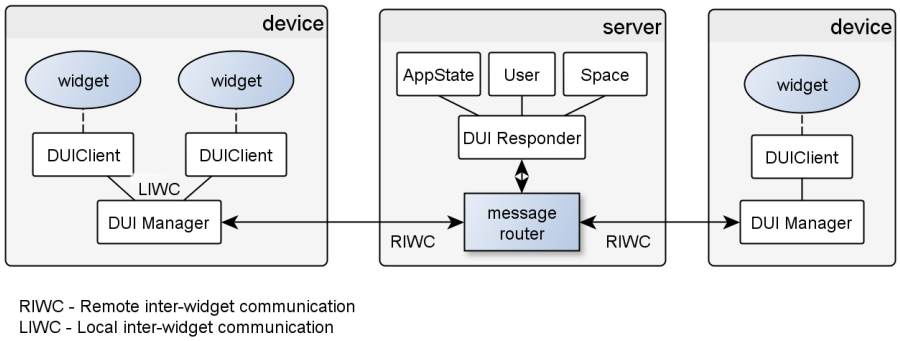


Fig. 4. Abstract architecture of the DUI framework

4.3 Widget Migration

By using a widget approach, the dynamic transition of UI components from desktop to mobile devices is simpler. Widgets resemble mobile device screen sizes by design. Rendering a widget on smartphone or a tablet only requires adaptation of the widget containing element. The framework provides primitives for continuous and non-continuous migration. The former preserves the widget state in the framework runtime, whereas the latter recreates the widget on the new location.

Considering failover, since mobile devices can go offline unexpectedly, widgets can become inactive. If failover happens during a continuous migration, the last known state is restored. The choice of migration type is left to the widget (application) developer to suit his needs. Moreover, the developer can design state persistence on at regular time intervals to ensure higher reliability. The DUI Responder considers a widget to be inactive if it cannot find an active device displaying the widget. Different procedures are provided to inactive widgets and active widgets. Figure 5 illustrates the case of continuous migration. When a DUI Manager initiates a widget migration on any device, the DUI Responder looks for the widgets on all devices of the requesting user. If the widget is found to be inactive, the DUI Responder switches the widget location from no device or an inactive device to the migration target device. Then, it sends out a message to perform the migration procedure on all DUI Managers.

During continuous migrations, widget state is stored right before migration. Thus, we ensure that widget state is preserved in cases of failure during continuous migration. The widget can retrieve state as a snapshot for continuing the task. DUI-supported widgets can be either inactive or active. DUI Manager tries to restore the state for inactive widgets and guarantees the continuity for active widgets. For inactive widgets, the steps are the same as the non-continuous migration of inactive widgets, except that DUI Manager sends the last stored state of the widget.

For continuous migration of active widgets, DUI Manager asks the widget's DUI Client to collect the widget state for the incoming migration. On receiving the command for migration, DUI Manager on the source device informs the DUI Client to prepare the widget removal. DUI Manager on the target device extracts information from the command. DUI Client is then guided by DUI Manager to run several steps to finish the migration.

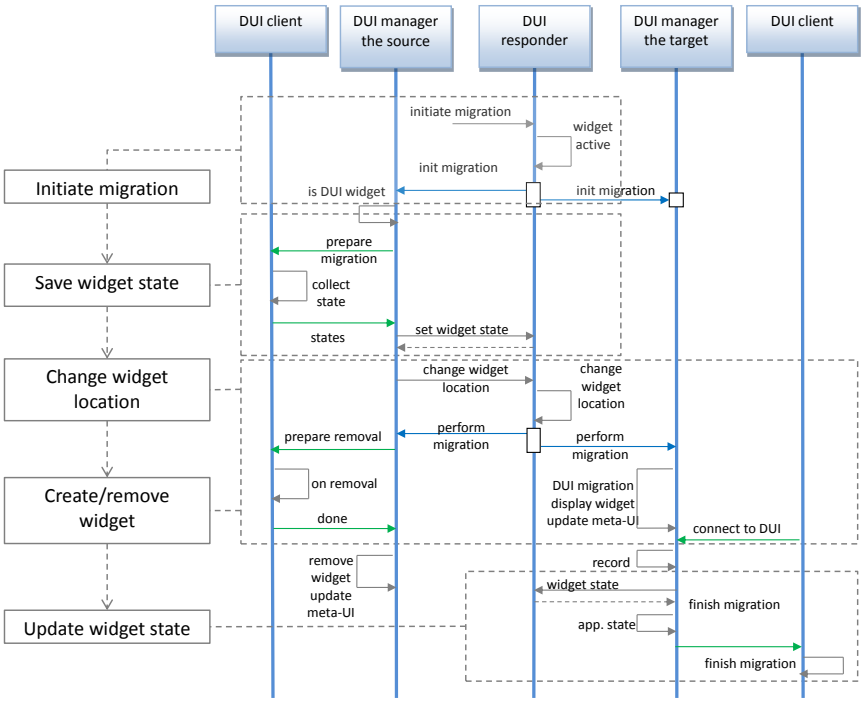


Fig. 5. Sequence diagram for continuous migration of active widgets

5 Implementation

The implementation of the DireWolf framework builds upon the Open Source Java-based ROLE SDK[a], including a platform for hosting and managing Widget-based Web applications as described in Section 3. As basic widget engine, the ROLE platform employs the OpenSocial [22] container Apache Shindig[b]. On top of Shindig, the platform implements a set of RESTful services for *user management* and *personal and collaborative widget space management*. It should be noted that the space concept is currently standardized in the OpenSocial 3.0 specification. Consequently, it will be implemented in Shindig and will possibly become part of other Shindig-based widget platforms such as Apache Rave[c]. Furthermore, the platform supports secure authentication and authorization by employing OpenID and OAuth. A real-time service realizes the integration with an XMPP [23] server providing support for multi-user chat conversations in widget spaces and publish-subscribe support for remote IWC. Associations between modules are realized by injection. For our work we strongly employ IWC, using HTML5 Web Messaging [24] for local IWC. An additional feature of our complete IWC approach includes remote communication between widgets across different browser instances and users [21] using the XMPP protocol [23] and its publish-subscribe extension [25]. We use both forms of IWC as a carrier for message exchange between different parts of our DUI framework within and across devices.

[a] `http://sourceforge.net/projects/role-project/`
[b] `http://shindig.apache.org/`
[c] `http://rave.apache.org`

On client side, the platform provides an AJAX browser frontend based on HTML, CSS, JavaScript, and jQuery.[d] For client-side real-time support the ROLE platform employs strophe.js,[e] a robust XMPP library for JavaScript including support for XMPP over WebSocket [26] in modern browsers. Widget spaces are used as context for IWC. In collaboration with user and space management services, the platform real-time service manages one dedicated publish-subscribe channel per space for IWC including whitelist-based access control. On client side, every widget space is instrumented with a DUI Manager including an *IWC proxy*, which routes outgoing IWC messages to the affiliated XMPP server via the strophe-based XMPP connection and incoming messages to all widgets in the space via HTML5 Web Messaging [24]. Widgets can be equipped with IWC support by simply importing a small *IWC client* library and implementing functions for publishing and processing IWC messages. The DUI Client library extends the plain IWC library by a set of functions related to storage and retrieval of internal widget state.

Given that many technical prerequisites for DireWolf were already fulfilled by the ROLE platform, we chose an integration approach. In its current version, DireWolf is an extension of the existing ROLE platform and its components. The DUI Responder is realized as an additional RESTful service for managing device migration-specific data such as personal device lists, device profiles, and user and space-related application states, including widget state. Client side components such as DUI Manager and DUI Client communicate application state and initiate widget migration by simple HTTP requests to the DUI Responder, which in turn controls the synchronization process and initiates real-time synchronization necessary for migration. All migration-related communication between individual components (Message Router, DUI Manager, DUI Clients) is handled via ROLE IWC over a separate publish-subscribe channel to avoid interference with regular developer-defined IWC messages.

For convenient control of widget distribution and device registration DireWolf provides a set of user interface components as frontend to the DUI Manager. Figure 6 shows the main component integrated into the side panel of a widget space's view in the overall ROLE platform user interface . The upper *Device Manager* button bar provides shortcuts to a device manager console for personal device management including detailed configuration and debugging options. The *Current Device* resp. *Remote Devices* sections list all widgets displayed on the current device resp. remote devices along with device connectivity. In the example in Figure 6, the current widget space contains six widgets, distributed to four devices with different profiles (PC, iPad, iPhone and Mac). Only two devices are currently active, indicated by the green circle next to the device name. Thus, only five widgets are currently visible. One widget was previously migrated to the user's iPhone, which is currently disconnected, indicated by a grey marker. By using drag and drop, widgets can be (re-)distributed between active devices. The user can simply drag a widget handler in the sidebar panel and drop on another device placeholder. The migration of the widget is then automatically triggered. At runtime, the user can create arbitrary distribution of widgets by migrating them to available devices.

[d] http://jquery.com/
[e] http://strophe.im/strophejs

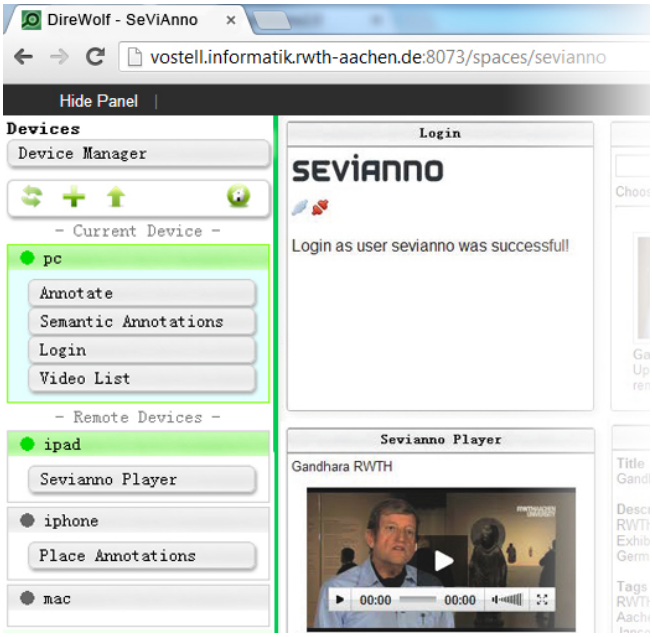


Fig. 6. DUI manager user interface in a widget space sidebar panel

5.1 WebRTC for Peer-to-Peer-Based Message Exchange

We are currently investigating to use the recent *Web Real-Time Communication (WebRTC)* [12, 13] draft to allow active devices to establish a peer-to-peer session to connect with each other. WebRTC allows direct data exchange between two browser instances without an intermediary server and across firewalls. Although current use cases from the draft are still focused on establishing media channels for exchanging audio and video streams, it includes the *DataChannel* interface for sending data bytes on the underlying UDP session, while preserving the reliability of TCP. In the case of DireWolf, XMPP-based remote IWC can be substituted by DataChannel links to bridge the devices of a user.

Additionally, we envision to offload centralized message routing logic from the server to the endpoints. In comparison to our current architecture (cf. Figure 4) the Message Router is now included on client side. To avoid a fully connected peer-to-peer mesh between the devices, one dedicated *Message Relay* is introduced in a single browser instance. The Message Relay is chosen by the system based on the connected device profiles. Hereby, a desktop system with stable network connection takes priority over smartphones with scarce battery resources. The Message Relay establishes and maintains WebRTC connections to other devices for relaying control messages from the server-side DUI Responder. The server component is still needed for managing widget application state.

To leverage the existing implementation and keep the use of peer-to-peer transparent to the widgets, we maintain the XML based communication semantics of the lower layer and only exchange the IWC's low-level WebSocket connection to the XMPP server by WebRTC DataChannel links. As WebRTC is relying on a signaling server to coordinate the connection establishment for synchronously negotiating holes in the firewall, all devices still need to open

an XMPP connection to the DUI Responder upon start. The setup is then performed by the Message Relay.

6 Evaluation

In this section, we present the setup of and results from our evaluation experiments which investigated the applicability and usability of the distribution of widget-based user interfaces in Web applications across different personal user devices. The evaluation of the DUI framework is divided into two parts. First, we measured and analyzed the technical properties of the widget migration operation. Second, we conducted an extensive user study for assessing the usability of the DireWolf framework. The evaluation targeted to measure the impact of the chosen technical framework upon the user experience, as well as to measure the user preferences and satisfaction, keeping into account the contrast to traditional Web applications. The methods used in the usability study try to discover the relation between different input and output possibilities and different devices, as well as the usage of more devices for achieving a customized personal computing environment, where users can interact with complex applications across multiple devices.

6.1 Widget Migration Performance

The migration component of the DireWolf framework was tested in a wireless local area network; a common environment at home or in office where a user would use DUIs. The ping latency of the network (6 ms) was considered negligible. Two setups were considered. The first setup measured migration between two desktop machines (Mac OS, Windows 7), using the Google Chrome browser (version 23). The second setup measured migration between desktop machines and an iPad 1 with iOS 5.0, using the Safari Web browser.

Tests were conducted with widgets with simple functionality, measuring the time between two consecutive migrations across two devices. In order to avoid noise induced by local time inconsistencies between devices, a reverse operation was automatically executed after initial migration, and total round-trip time was recorded. For consistency reasons, two kinds of migrations – simple migration (non-state-preserving) and continuous migration (state-preserving) – were evaluated. Round trip times for 100 migrations (i.e. 50 rounds) were measured.

Overall, our prototype achieved good performance results. For a blank widget, migration lasted on average M = 0.36 s (SD = 0.05 s). Continuous migration requires two more steps than a simple migration, i.e. storing widget state and rendering the widget with the Apache Shindig rendering engine. Average time for continuous migration between the MacBook and the desktop computer was M = 1.31 s (SD = 0.15 s). Due to the hardware differences, MacBook and iPad combination yielded higher average migration time (M = 2.06 s, SD = 0.22 s).

By decomposing the time necessary for the migration and observing the interval needed by each component of our framework, the results show that the initiation and the widget rendering process take more time than the migration itself. The Shindig server's JavaScript library loading and the widget rendering steps require approximatively 69% of the time. In contrast, the loading time needed by the DUI components is less than 25% of the overall time.

6.2 Usability Analysis

In this section we report on two user studies in which the participants performed the tasks distribution of user interfaces as well as widget transitions across various devices. The 25 participants were students or young researchers, studying in different domains at university level. At first, participants were familiarized with the DireWolf framework, the concept of Web widgets and spaces. Next, the multi-device distribution of user interfaces was demonstrated on a simple widget-based Web application. Finally, the users were asked to perform tasks in two subsequent experiments. The first one tested the user preferences for performing certain activities using various widget types and devices. The second studied user performance and experience with a complex Web application for video annotation with and without distribution of the user interface. The user studies were arranged in individual sessions, after which each user had to complete a usability questionnaire. The answers of the participants were collected and compared. In addition, user interactions with the prototype were measured.

6.2.1 Widget Preferences

In this experiment we assumed that users can perform certain tasks like painting or typing better or worse, depending on which type of device like touch-screen or desktop computer they are working on. Three different widgets were involved in this experiment: a *painting widget*, a *text input widget* and a *map widget*. The participants were asked to draw a given picture on a canvas using both a touch-screen device and a desktop PC. The drawing performance in terms of time needed to finish the drawing was measured. This type of widget was used because the painting action requires accurate targeting and continuous movements on the screen. In the text input widget case, the participants were asked to type a given text. The times for finishing text input using on-screen vs. physical keyboards were recorded. This widget assesses how well the on-screen mimicks the physical keyboard and how people react on these two input modalities. In the map widget case, the participants were told a set of local sights equally known by all persons. The time for locating these places on a map was recorded. This task required a different set of interactions, depending on the device used. For the touch-screen device, multi-touch gestures were involved like swiping and two-finger zoom. For the mouse supported device, actions such as wheel zooming and dragging were involved. The map widget assessed how people react on these two sets of interactive operations. Upon task completion, participants stated that they preferred device/widget combinations for different tasks.

Although users perform better on touch screens for drawing (mouse supported device: M = 19.4 s, SD = 5.4 s; touch screen device: M = 15.1 s, SD = 4.2 s), 67% of users preferred the mouse device. The typing task revealed that 96% of users preferred hardware keyboard, indicating that the touch screen keyboard only provides an alternative, but not an optimal solution (performance results with hardware keyboard: M = 48.7 s, SD = 6.7 s; touch screen keyboard: M = 108.7 s, SD = 23.6 s). The map navigation task showed that although people perform better on mouse supported devices (mouse supported device: M = 39 s, SD = 9.4 s; touch screen M = 48.9 s, SD = 10.3 s), more than half of users think that touch screens exceed traditional desktop computer capabilities for this particular type of interaction.

The evaluation results of this study support the initial assumptions about device preferences for different tasks. This implies that there is a need of widget distribution and migration. Overall, the preferences do not only depend on the performance, but also on the interaction style and the familiarity of the users. For the paint widget most users declared that they had never painted on touch screens before. The results obtained using the paint and the map widgets exemplify the mutual influence between the familiarity and style of interaction. Furthermore, the result of the text input is an example for the overwhelming influence of the performance factor.

6.2.2 DUIs for Complex Web Applications

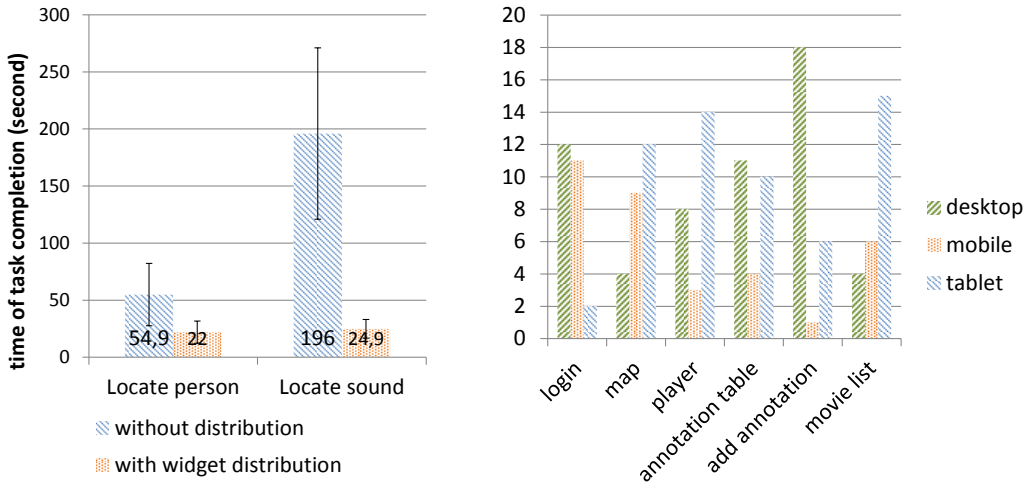
This part summarizes the second usability study, which evaluated the usefulness of the DireWolf framework for distributing the interface of a *complex Web application*. We validated the benefits of widget-based DUIs across devices, while the user performs a complex UI interaction task. This experiment used SeViAnno 2.0 (cf. Figure 1), our widget-based research prototype with a customizable and distributable user interface for semantic video annotations [27]. The application is composed of five widgets, fulfilling the following functionalities: video player, adding annotations, listing of annotations for display and navigation, annotating places using a map and listing all the available videos. A widget can migrate between devices without losing its current state. Thus, after distributing the SeViAnno 2.0 widgets to multiple devices, the widgets continue their operation while the whole application preserves its integrity and functionality. For example, after migrating the video player widget it resumes playing without any interruptions.

In our study, SeViAnno 2.0 used a desktop PC to display a full screen video player, an iPad to enable video navigation using the list of existing annotations and an iPhone to display the map widget for spatial annotations. Users were asked to watch a video clip and locate the time point of a certain event. Using this setting, we demonstrated the usefulness of distributed interfaces by showing that content can be found faster. Two ways in which a user could locate the right time point in a given time frame were considered. One way was to randomly browse through the video using the video seek bar. The second way was to search information in the video annotations list, where content about persons, events, places, objects, etc. was available. In the evaluation scenario, the first search method is used when the user does not distribute the widget and watches the video clip in full screen mode. With a full screen display, the second search method is available only if the widget is located on another device.

Considering a single video, the first task was to locate the occurrence of a known person in the video and the second task to locate the background voice that introduces a person in the video. The time spent for each of the two search tasks is depicted in Figure 7a. The data recorded is the time that the user spends on locating the right time point in the video.

After the experiments, users were free to rearrange the widget distribution and to perform other tasks. One of the observations at this stage was that due to user preference on single widgets and the multi-platform compatibility of the widget implementation, different users can have different preferences on the distribution of the widget set (see Figure 7b).

The results show that users found the information faster with the help of the annotations widget. Considering the deviation, the search was more reliable with the help of the list of



(a) Time for task completion with and without distribution

(b) User preferences of widget distribution

Fig. 7. Results of the evaluation using SeViAnno 2.0

annotations. In the results, a high deviation implies that the task is done by chance. This became more evident as the information was harder to locate. Under the condition that the users watched the video in full screen mode, such results imply that the distribution among multiple devices improves the usability of complex Web applications.

Figure 7b shows the preferences on how users distributed the SeViAnno widgets. As it can be observed, the users preferred to place the interactive widgets on the tablet. Users chose the desktop device for widgets involving text input, as observed in the previous experiments. The mobile phone was less used by users, because of the small screen size. This also implies that users do distribute the widgets based on their preferences.

Concerning the overall user experience, the results show that distributing user interfaces brings a good improvement for the usability and that this is a helpful feature (average 4.1/5). Users would consider DUIs in the future, depending on the type of application (average 3.44/5). The DUI was considered easy to use, with an intuitive migration and an easy device management. The migration time was not considered fast, but users rated it as acceptable (average 3.48/5). Finally, we have also gathered general user feedback, using a section at the end of our questionnaire for general comments. For the implementation of the complex application, users expected better widget optimization for mobile devices. Related to the DUI framework, users expected a reduced amount of side panel elements (e.g. information about widgets, users) when there are many widgets in the space. Furthermore, different interaction capabilities for the widget migration have been mentioned, such as drag-and-drop of the widget itself to the target device. We consider these comments for further prototype improvements. Furthermore, regarding our user study, the context switch for widget distribution should also be considered, as switching between devices is dependent on the type of devices and their functions.

7 Limitations

For this work we restricted our focus to a single-user multi-device setup. In such a setting, it might appear that security does not play a major role. The complete setup is running on a local network. All devices must be registered to the framework explicitly. All XMPP traffic managing the inter-widget communication between devices is secured, e.g. with TLS. However, prior experience with widget-based approaches has shown that even local forms of inter-widget communication can pose serious security risks. In general, widgets used in DireWolf must be considered as third-party products. As such, their internals remain unknown to most users. Part of such internals is the widget's IWC interface, i.e. which messages it emits and to which messages it reacts accordingly. In DireWolf, the user has no explicit control and awareness of the data flows between widgets. Without such control, third party widgets with intentionally or unintentionally malicious behaviour are not hindered to carry out their routines. A typical example is a widget capable of receiving intents to use paid services, e.g. sending an SMS [20]. Combined with another widget emitting such intents without explicit consent, the user will be held liable for the costs incurred. The situation would be even worsened with multiple users collaborating remotely. As countermeasures, future versions of DireWolf should introduce explicit mechanisms for controlling and staying aware of the particular communication between widgets, thus resembling a hybrid UI mashup (cf. [20]). In a single-user setting, a user should be enabled to manage trust for an individual widget in terms of allowing resp. forbidding its participation in inter-widget communication. In a multi-user setting, a user should be enabled to additionally manage trust to agents and - in case of trust violations - block communication with those agents. Although TLS is the common standard technique for message encryption in XMPP, we would prefer end-to-end encryption (e.g. OTR), as currently actively discussed in the XMPP community.

8 Conclusions and Future Work

In this paper, we aim to solve the lack of dynamic interactive environments based on Web technologies which can take advantage of the various personal devices used by an individual. We provide a framework that can facilitate user interactions on a federation of personal computing devices by making use of distributed user interfaces. We follow a widget-based approach to encapsulate UIs and application functionalities, which benefits Web developer communities already familiar with this programming model. Apache Rave and Shindig are examples of such open-source communities. Since widgets can be grouped, shared, reused and personalized, our approach ensures a unique user experience with DUI applications. The DireWolf framework also provides features for distributing and migrating widgets, while hiding the complexity of device awareness, inter-widget communication and session mobility. As the performance evaluation indicated, the framework adds only a small overhead to the overall widget rendering process. The DireWolf framework paves the way for many interesting experiments. Our user studies confirmed the assumptions that users have certain preferences for widget/device combinations which are influenced by the performance factor and task specificity. We tested additional interaction modalities within the semantic video annotation application illustrated in Figure 1, which confirmed the usefulness of the DUI approach for complex Web applications.

We envision our framework in the domains of technology-enhanced learning, smart TV and public display interaction. On system level, Section 5.1 sketches our ongoing work on integrating the emerging WebRTC API for near real-time browser-to-browser communication. This will eliminate the need of an intermediary messaging server and will address security and privacy issues in message exchange across devices and users. To underline our efforts to build standard-based solutions and extend existing standards where needed, we have submitted an XMPP extension that has since reached experimental status. On design level, we are taking additional steps towards enhancing the overall user experience. For instance, we are going to apply responsive Web design principles to overcome usability issues with widget rendering on heterogeneous mobile devices, identified during the evaluation sessions. One further potential enhancement is to implement a more natural, gesture-based shifting of widgets across devices. On conceptual level, we will extend DireWolf to support multi-device multi-user collaboration, as a next step beyond the personal multi-device distributed computing environment. We also plan to investigate what combinations of devices and widget types would be the most relevant for distribution.

Acknowledgements

The research leading to these results has received funding from the European Commission's Seventh Framework Programme (FP7/2007-2013) under grant agreements no 231396 - Responsive Open Learning Environments (ROLE) project and no 318209 - Learning Layers: Scaling up Technologies for Informal Learning in SME Clusters and the Excellence Initiative of German National Science Foundation (DFG) within the research cluster Ultra High-Speed Mobile Information and Communication (UMIC). We thank Ke Li for his framework implementation.

References

1. J. J. Lòpez-Espin, J. A. Gallud, E. Lazcorreta, A. Peñalver, and F. Botella. A Formal View of Distributed User Interfaces. In *Proceedings of the Distributed User Interfaces CHI 2011 Workshop*, pages 97–100, Vancouver, BC, Canada, 2011. University of Castilla-La Mancha, Spain.
2. C. Vandervelpen, G. Vanderhulst, K. Kris Luyten, and K. Coninx. Light-Weight Distributed Web Interfaces: Preparing the Web for Heterogeneous Environments. In *Proceedings of the 5th International Conference on Web Engineering*, volume 3579 of *LNCS*, pages 197–202, Sydney, Australia, 2005. Springer-Verlag Berlin.
3. M. Blumendorf, D. Roscher, and S. Albayrak. Distributed User Interfaces for Smart Environments: Characteristics and Challenges. In *Proceedings of the Distributed User Interfaces CHI 2011 Workshop*, pages 25–28, Vancouver, BC, Canada, 2011. University of Castilla-La Mancha, Spain.
4. J. Vanderdonckt. Distributed User Interfaces: How to Distribute User Interface Elements Across Users, Platforms, and Environments. In *Proceedings of the XI Congreso Internacional Interacción Persona-Ordenador*, INTERACCIÓN 2010, pages 20–32, Valencia, Spain, 2010. ACM.
5. E. Bogdanov, C. Salzmann, and D. Gillet. Contextual Spaces with Functional Skins as OpenSocial Extension. In *Proceedings of the Fourth International Conference on Advances in Computer-Human Interactions (ACHI 2011)*, pages 158–163, Gosier, Guadeloupe, France, 2011.
6. F. Daniel, S. Soi, S. Tranquillini, F. Casati, C. Heng, and L. Yan. Distributed Orchestration of User Interfaces. *Information Systems*, 37(6):539–556, 2012.
7. G. Ghiani, F. Paternò, and C. Santoro. On-demand Cross-Device Interface Components Migration.

In *Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI '10)*, pages 299–308, Lisabon, Portugal, 2010. ACM Press.

8. L. Baillie, R. Schatz, R. Simon, H. Anegg, F. Wegscheider, G. Niklfeld, and A. Gassner. Designing Mona: User Interactions with Multimodal Mobile Applications. In *Proceedings of 11th International Conference on Human-Computer Interaction (HCI International)*, pages 22–27, Las Vegas, NV, USA, 2005. Lawrence Erlbaum Associates.
9. K. Luyten and K. Coninx. Distributed User Interface Elements to support Smart Interaction Spaces. In *Proceedings of the Seventh IEEE International Symposium on Multimedia*, ISM '05, pages 277–286, Irvine, CA, USA, 2005. IEEE Computer Society.
10. P. G. Villanueva, R. Tesoriero, and J. A. Gallud. Proxywork: Distributing User Interface Components of Web Applications. In *Proceedings of the 3rd Workshop on Distributed User Interfaces: Models, Methods and Tools, DUI 2013 in conjunction with EICS 2013 Conference*, pages 58–61, London, UK, 2013.
11. J. Melchior, D. Grolaux, J. Vanderdonckt, and P. van Roy. A Toolkit for Peer-to-peer Distributed User Interfaces: Concepts, Implementation, and Applications. In *Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 69–78, Pittsburgh, PA, USA, 2009. ACM Press.
12. C. Jennings, T. Hardie, and M. Westerlund. Real-time communications for the Web. *IEEE Communications Magazine*, 51(4):20–26, 2013.
13. A. Bergkvist, D. C. Burnett, C. Jennings, and A. Narayanan. WebRTC 1.0: Real-time Communication Between Browsers. Working draft, W3C, 2013.
14. H.-H. Chu, H. Song, C. Wong, S. Kurakake, and M. Katagiri. Roam, a Seamless Application Framework. *Journal of Systems and Software*, 69(3):209–226, 2004.
15. A. Alapetite. Dynamic 2D-barcodes for Multi-Device Web Session Migration Including Mobile Phones. *Personal Ubiquitous Computing*, 14(1):45–52, 2010.
16. F. Paternò, C. Santoro, and A. Scorcia. User Interface Migration Between Mobile Devices and Digital TV. In *Proceedings of the 2nd Conference on Human-Centered Software Engineering and 7th International Workshop on Task Models and Diagrams*, pages 287–292, Pisa, Italy, 2008. Springer-Verlag Berlin.
17. B. A. Myers. Using Handhelds and PCs Together. *Communications of the ACM*, 44(11):34–41, 2001.
18. F. Daniel, S. Soi, S. Tranquillini, F. Casati, H. Chang, and Y. Li. MarcoFlow: Modeling, Deploying, and Running Distributed User Interface Orchestrations. In *Proceedings of the 8th International Conference on Business Process Management Demo Track*, BPM 2010, pages 23–27, Hoboken, NJ, USA, 2010. Springer.
19. J. S. Pierce and J. Nichols. An Infrastructure for Extending Applications' User Experiences Across Multiple Personal Devices. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology (UIST '08)*, pages 101–110, Monterey, CA, USA, 2008. ACM Press.
20. S. Wilson, F. Daniel, U. Jugel, and S. Soi. Orchestrated User Interface Mashups Using W3C Widgets. In *Proceedings of the 11th International Conference on Current Trends in Web Engineering*, ICWE'11, pages 49–61, Paphos, Cyprus, 2011. Springer-Verlag.
21. S. Govaerts, K. Verbert, D. Dahrendorf, C. Ullrich, M. Schmidt, M. Werkle, A. Chatterjee, A. Nussbaumer, D. Renzel, M. Scheffel, M. Friedrich, J. L. Santos, E. Duval, and E. L.-C. Law. Towards Responsive Open Learning Environments: the ROLE Interoperability Framework. In *Proceedings of the 6th European Conference on Technology Enhanced Learning: Towards Ubiquitous Learning*, EC-TEL'11, pages 125–138, Palermo, Italy, 2011. Springer-Verlag.
22. OpenSocial and Gadgets Specification Group. OpenSocial Specification 2.5.0. http://opensocial-resources.googlecode.com/svn/spec/2.5/. Online: last accessed March 2013.
23. P. Saint-Andre. RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. Technical report, XMPP Standards Foundation, 2011.
24. I. Hickson. HTML5 Web Messaging. Working draft, W3C, 2011.
25. P. Millard, P. Saint-Andre, and R. Meijer. XEP-0060: Publish-Subscribe Version 1.13, Draft.

Technical report, XMPP Standards Foundation, 2010.
26. I. Hickson. The WebSocket API. Editor's draft, W3C, 2013.
27. Y. Cao, D. Renzel, M. Jarke, R. Klamma, M. Lottko, G. Toubekis, and M. Jansen. Well-Balanced Usability and Annotation Complexity in Interactive Video Semantization. In *Proceedings of the 4th International Conference on Multimedia and Ubiquitous Engineering (MUE 2010)*, pages 1–8, Cebu, Philippines, 2010. IEEE.