DireWolf - Distributing and Migrating User Interfaces for Widget-Based Web Applications

Dejan Kovachev, Dominik Renzel, Petru Nicolaescu, and Ralf Klamma

Advanced Community Information Systems (ACIS) Group,
RWTH Aachen University,
Ahornstr. 55, 52056 Aachen, Germany
{kovachev,renzel,nicolaescu,klamma}@dbis.rwth-aachen.de,
http://dbis.rwth-aachen.de

Abstract. 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 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 collaborative 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. 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.

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.

F. Daniel, P. Dolog, and Q. Li (Eds.): ICWE 2013, LNCS 7977, pp. 99–113, 2013.

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. 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 [4,5] 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
- facilitation of extended multi-modal real-time interactions on a federation of personal computing devices
- provision of continuous state-preserving widget migration

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 is detailed in the next sections.

The rest of the paper is structured as follows. In Section 2, relevant literature related to our approach is presented. 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. Section 5 provides implementation details. Evaluation results are discussed in Section 6. Section 7 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. [6] 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 [2,7,8] 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. But reusability and extensibility of sub-services/components are major shortcomings in these approaches. A new UI schema needs to be fixed for a complete application. Sub-service definitions cannot be separated. Consequently, the services of an application cannot be ported with ease. 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.

Dynamic DUIs should support runtime component migration. Necessary steps for a successful migration are presented in the Roam project [9]. 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. [10] migrate Web sessions across mobile devices using 2D-barcodes captured by cameras. A dedicated State Mapper is also developed in [11] 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 [12] 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. [13] 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 [14,5] 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 [15] 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* 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* [4] 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-widget Communication (IWC)*. With such a usually publish-subscribe-based mechanism, messages can be broadcasted from any widget and possibly dispatched by other widgets, thus allowing the orchestration [16] and tighter integration of multiple widgets to complete applications. 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 [17]. 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.

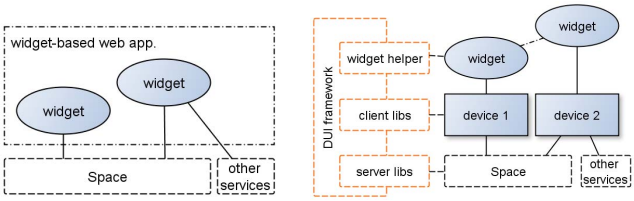
Figure 2a depicts the initial setting from which this work departed. In the following section we elaborate on the extensions contributed by our DUI framework in detail, thus leading to the situation in Figure 2b.

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.

The DUI framework is involved in every layer of the widget-based Web application. As shown in Figure 2b, 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



(a) Traditional (non-distributed) approach (b) DUI approach

Fig. 2. Widget-based Web applications

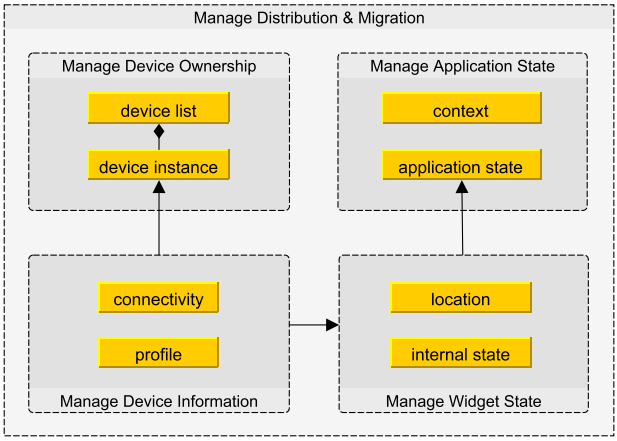


Fig. 3. Requirements to a dynamic widget-based DUI framework

smartphones and tablets. The use cases focus on creating, getter/setter and operating on resources (widgets).

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 for a DUI framework, grouped into four interrelated categories: *device information*, *device ownership*, *distribution & migration*, *application state* and *widget handling*.

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

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/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.

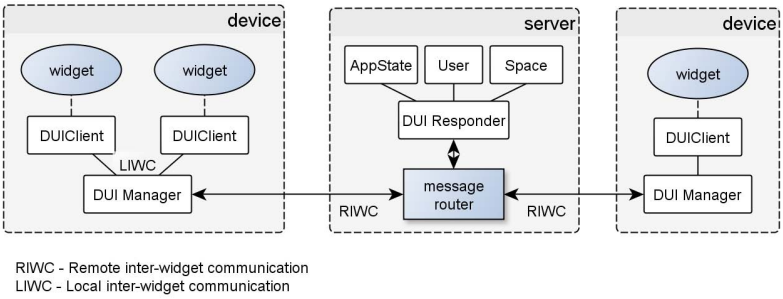


Fig. 4. Abstract architecture of the DUI framework

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.

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.

Considering the failover, since mobile devices can go offline unexpectedly, widgets can become inactive. 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 saved right before 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 saved 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.

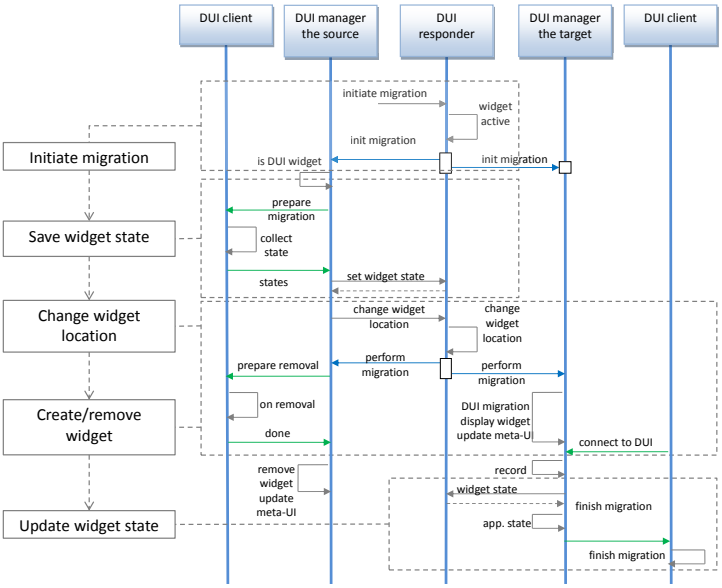


Fig. 5. Sequence diagram for continuous migration of active widgets

5 Implementation

The implementation of the DireWolf framework builds upon the Open Source Java-based ROLE SDK[1] 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 standard OpenSocial [18] container Apache Shindig[2]. 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[3]. Furthermore, the platform supports secure authentication and

[1] `http://sourceforge.net/projects/role-project/`
[2] `http://shindig.apache.org/`
[3] `http://rave.apache.org`

authorization by employing OpenID and OAuth. A real-time service realizes the integration with a standard XMPP [19] 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 [20] for local IWC. An additional feature of our complete IWC approach includes remote communication between widgets across different browser instances and users [17] using the XMPP protocol [19] and its publish-subscribe extension [21]. We use both forms of IWC as a carrier for message exchange between different parts of our DUI framework within and across devices.

On client side, the platform provides an AJAX browser frontend based on HTML/JavaScript/CSS and jQuery[4]. For client-side real-time support the ROLE platform employs strophe.js[5], a robust XMPP library for JavaScript including support for XMPP over WebSocket [22] 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 [20]. 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. 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

[4] `http://jquery.com/`
[5] `http://strophe.im/strophejs`

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

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.

6 Evaluation

The focus of our experiments was to research how distributing widget-based user interfaces in Web applications across different personal user devices can be achieved. In this section, we briefly present performance evaluation results regarding widget migration.

The migration component of the DireWolf framework was tested on a wireless local area network, simulating the home or office conditions. The ping latency of the network (of 6ms) 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. Average migration time for simple migration was around 362.6 ms for a "hello world" widget with a standard deviation of 48.9 ms. Continuous migration requires two more steps than 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 1305 ms, with a standard deviation of 147.2 ms. The results show higher average migration time between the MacBook and the iPad, i.e. an average time of 2069 ms and the standard deviation of 222.6 ms. This is due to the hardware differences and the time needed to load all the dependencies. 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.

The presented evaluation is limited to technical properties of the widget migration feature. However, we conducted an extensive user study for assessing the usability of the DireWolf framework, which due to space limitations could not be discussed in this paper. In addition, DireWolf is currently being tested on a bigger range of devices. Even though DireWolf has been derived from the existing ROLE Widget SDK, as described in Section 5, it is not yet included into an existing official SDK release. Encouraged by the small overhead and latency that the framework introduces, the next step is to integrate DireWolf into the future versions of the SDK.

7 Conclusions and Future Work

In this paper, we try to leverage 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. Furthermore, we believe that a widget-based approach to encapsulate UIs and application functionalities 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 unique user experiences with DUI applications. Our framework also provides features for distributing and migrating widgets, at the same time hiding the complexity of device awareness, communication and session mobility. As initial evaluation indicates, the framework adds only small overhead to the overall widget rendering process.

The framework we present here paves the way for many interesting experiments. We are already testing complex interaction modalities within the semantic

video annotation application illustrated in Figure 1. Furthermore, we envision our framework in the domains of technology-enhanced learning and interactive smart television. We also consider using the emerging WebRTC project[6] for real-time browser-to-browser communication without a server intermediate. As a next step beyond the personal multi-device distributed computing environment, we will extend DireWolf to support multi-device multi-user collaboration. Further research must address security and privacy issues in message exchange across devices and users. We are committed to open source development and we aim to integrate the IWC and DireWolf within an open source project, such as Apache Rave. We plan to provide tutorials, Web casts and code snippets with intention to form a sustainable developer community around our solution.

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. Lòpez-Espin, J.J., Gallud, J.A., Lazcorreta, E., Peñalver, A., Botella, F.: A Formal View of Distributed User Interfaces. In: Distributed User Interfaces CHI 2011 Workshop, University of Castilla-La Mancha, Spain, pp. 97–100 (2011)
2. Vandervelpen, C., Vanderhulst, G., Luyten, K., Coninx, K.: Light-Weight Distributed Web Interfaces: Preparing the Web for Heterogeneous Environments. In: Lowe, D.G., Gaedke, M. (eds.) ICWE 2005. LNCS, vol. 3579, pp. 197–202. Springer, Heidelberg (2005)
3. Blumendorf, M., Roscher, D., Albayrak, S.: Distributed User Interfaces for Smart Environments: Characteristics and Challenges. In: Distributed User Interfaces CHI 2011 Workshop, University of Castilla-La Mancha, Spain, pp. 25–28 (2011)
4. Bogdanov, E., Salzmann, C., Gillet, D.: Contextual Spaces with Functional Skins as OpenSocial Extension. In: The Fourth International Conference on Advances in Computer-Human Interactions, ACHI 2011, pp. 158–163 (2011)
5. Daniel, F., Soi, S., Tranquillini, S., Casati, F., Heng, C., Yan, L.: Distributed Orchestration of User Interfaces. Information Systems 37(6), 539–556 (2012)
6. Ghiani, G., Paternò, F., Santoro, C.: On-demand Cross-Device Interface Components Migration. In: Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI 2010), pp. 299–308. ACM Press (2010)
7. Baillie, L., Schatz, R., Simon, R., Anegg, H., Wegscheider, F., Niklfeld, G., Gassner, A.: Designing Mona: User Interactions with Multimodal Mobile Applications. In: Proceedings of 11th International Conference on Human-Computer Interaction (HCI International), pp. 22–27. Lawrence Erlbaum Associates (2005)

[6] `http://www.webrtc.org/`

8. Luyten, K., Coninx, K.: Distributed User Interface Elements to support Smart Interaction Spaces. In: Proceedings of the Seventh IEEE International Symposium on Multimedia, ISM 2005, pp. 277–286. IEEE Computer Society (2005)
9. Chu, H.H., Song, H., Wong, C., Kurakake, S., Katagiri, M.: Roam, a Seamless Application Framework. Journal of Systems and Software 69(3), 209–226 (2004)
10. Alapetite, A.: Dynamic 2D-barcodes for Multi-Device Web Session Migration Including Mobile Phones. Personal Ubiquitous Computing 14(1), 45–52 (2010)
11. Paternò, F., Santoro, C., Scorcia, A.: User Interface Migration Between Mobile Devices and Digital TV. In: Forbrig, P., Paternò, F. (eds.) HCSE/TAMODIA 2008. LNCS, vol. 5247, pp. 287–292. Springer, Heidelberg (2008)
12. Myers, B.A.: Using Handhelds and PCs Together. Communications of the ACM 44(11), 34–41 (2001)
13. Melchior, J., Grolaux, D., Vanderdonckt, J., van Roy, P.: 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, pp. 69–78. ACM Press (2009)
14. Daniel, F., Soi, S., Tranquillini, S., Casati, F., Chang, H., Li, Y.: MarcoFlow: Modeling, Deploying, and Running Distributed User Interface Orchestrations. In: Proceedings of the 8th International Conference on Business Process Management Demo Track, pp. 23–27. Springer (2010)
15. Pierce, J.S., Nichols, J.: 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 2008), pp. 101–110. ACM Press (2008)
16. Zuzak, I., Ivankovic, M., Budiselic, I.: A Classification Framework for Web Browser Cross-Context Communication. CoRR abs/1108.4770 (2011)
17. Govaerts, S., Verbert, K., Dahrendorf, D., Ullrich, C., Schmidt, M., Werkle, M., Chatterjee, A., Nussbaumer, A., Renzel, D., Scheffel, M., Friedrich, M., Santos, J.L., Duval, E., Law, E.L.-C.: Towards responsive open learning environments: the ROLE interoperability framework. In: Kloos, C.D., Gillet, D., Crespo García, R.M., Wild, F., Wolpers, M. (eds.) EC-TEL 2011. LNCS, vol. 6964, pp. 125–138. Springer, Heidelberg (2011)
18. OpenSocial and Gadgets Specification Group: OpenSocial Specification 2.5.0, `http://opensocial-resources.googlecode.com/svn/spec/2.5/` (Online: last accessed March 2013)
19. Saint-Andre, P.: RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. Technical report, XMPP Standards Foundation (2011)
20. Hickson, I.: HTML5 Web Messaging. Working draft, W3C (2011)
21. Millard, P., Saint-Andre, P., Meijer, R.: XEP-0060: Publish-Subscribe Version 1.13, Draft. Technical report, XMPP Standards Foundation (2010)
22. Hickson, I.: The WebSocket API. Editor's draft, W3C (2013)