Tandem Browsing Toolkit: Distributed Multi-Display Interfaces with Web Technologies

Tommi Heikkinen[1], Jorge Goncalves[1], Vassilis Kostakos[1], Ivan Elhart[2], Timo Ojala[1]
[1]Department of Computer Science and Engineering
University of Oulu, Finland
{firstname.lastname}@ee.oulu.fi
[2]Faculty of Informatics
University of Lugano (USI), Lugano, Switzerland
{firstname.lastname}@usi.ch

ABSTRACT
We present the Tandem Browsing toolkit that allows developers to build multi-display and multi-user applications for pervasive displays with web technologies. Existing tools for this purpose either focus on user needs, rather than developer needs, or do not rely on open web standards. Our proxy-based toolkit allows developers to conceptualize, design and implement interfaces that orchestrate multiple devices in navigating through online content, without any modifications to user devices. We first describe the design and implementation of our toolkit, followed by a qualitative validation with web developers. Then we illustrate the functionality of the toolkit with three prototypes. We conclude with a discussion on the toolkit's characteristics and capabilities.

Categories and Subject Descriptors
H.5.2 [**Information Interfaces and Presentation**]: User Interfaces - *Graphical User Interfaces*.

General Terms
Design, Languages, Management.

Keywords
Multibrowsing; distributed user interfaces; shared navigation.

1. INTRODUCTION
As increasing numbers of mobile and fixed displays populate our environment, researchers have sought new ways for users to take advantage of these devices. Pervasive displays are inherently shared by their users and therefore can substantially benefit from applications that support multiple displays at once. By enabling multi-display applications, the diversity of devices and their form factors can be fully exploited. In addition, the collaborative use of applications can be substantially strengthened: while users use the same applications using different devices, the applications can support shared use by coordinating activities and synchronizing content between the users.

The currently available multi-display tools empower users to take control of their task and use multiple devices in a distributed user interface fashion [11]. For instance a user may choose to use a tablet pc to search for images and videos, but switch to the large TV in the living room to actually display those. Similarly, groups of users are able to better synchronize their activities by keeping track of each other's actions while executing a shared task.

The flexibility of the web, and the ease with which online content can be accessed and shared, has made web technologies an interesting approach for collaborative browsing involving multiple devices. However, most existing approaches have an important potential limitation: they focus on the user, not the developer. Most tools are built as ad-hoc extensions that turn web browsing a shared activity by simply synchronizing all the actions between the participating browsers. The focus on users also has another important consequence: it is hard for a developer to actually create content, online applications, or orchestrate an online experience taking advantage of multiple devices. We argue that the full potential of multi-display applications can be achieved only if the applications are designed from the ground up. For this to happen developers require tools, which are currently scarce.

In this paper we present the Tandem Browsing toolkit, a proxy-based toolkit for building multi-display applications and online experiences. The toolkit allows developers to declaratively define multipart web pages, i.e., pages that have multiple conceptual parts viewed on multiple devices in tandem. The toolkit uses a finite state machine metafile for building multipart pages and establishing stateful application flow (navigation sequence). The toolkit extends our previous work on the declarative layout management of the screen real estate of interactive public displays with web technologies [5, 12] by adding support for multi-display and multi-session applications.

2. RELATED WORK
Various web based tools for multi-display applications have been presented in literature. A popular approach has been to employ a proxy for controlling a browsing session spanning multiple devices. For example, Cabri *et al.* [2] presented a multi-device system where a proxy informs all devices in the session about the status and activities of other devices, and enables multiple users to chat with each other while browsing. Taking this idea further, Johanson *et al.* [7] described a multi-browsing framework that empowers users to continue their browsing session across multiple devices and synchronizing their activities via a tuplespace. For instance a user can start browsing on her desktop and then continue browsing on a tablet PC. In both frameworks clients have to have a specific browser plugin, which is not the case in our toolkit. Atterer *et al.* [1] went even as far as synchronizing cursor, scrolling movements and keyboard inputs across multiple devices with a web based system that does not require any plugins. However, their system focuses on end-user needs, and as such, does not provide support for designing applications specifically for multi-display scenarios.

Maekawa *et al.* [13] proposed a more modular system for collaborative browsing on mobile phones. Their system partitions

PerDis '14, June 03 - 04 2014, Copenhagen, Denmark
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2952-1/14/06..$15.00.
http://dx.doi.org/10.1145/2611009.2611026

web pages by analyzing their elements and the capabilities of the mobile clients. Users are allowed to define rules for each client regarding browsing across multiple devices. This approach results in the need for distributed rules configured by users on an ad-hoc basis, thus increasing flexibility at the expense of user burden.

Some studies have considered ways to empower developers in taking advantage of multiple devices in their applications. For example, the WebSplitter [4] defined "partial views" for designing content on multiple devices using specific XML pages. This approach allows multiple devices to render different parts of the same XML file based on decisions made upon design. Our toolkit also uses XML for defining pages with multiple parts. However, in our case the actual content is created using standard web pages familiar to web developers. Coles *et al.* [3] described a framework for coordinated web form filling that enables different representations of pages across multiple clients. The framework avoids the need for a stateful proxy by aggregating the page request from clients with the help of a stateless consolidation proxy. However, the framework requires a customized browser.

We also highlight that a number of web technologies are available to build applications where the content is synchronized between multiple clients. For instance, web sockets, Ajax and reverse Ajax, and Java applets allow developers to build pages where two or more clients can directly interact in real time. However, what these technologies lack is a way for clients to be synchronized and orchestrated in the way they move between pages.

In summary, developing multi-display applications with web technologies has been an active research topic for many years, but mostly for the benefit of end users. Most past work focuses on empowering end users to collaborate browsing over existing content and to find appropriate ways to utilize multiple devices for improving browsing experience. Interestingly, the focus on web developers has been rather limited so far − very few tools have been built to empower developers to specifically design applications for multi-display settings. This is the goal of our tandem browsing toolkit.

3. TANDEM BROWSING

The objective of the Tandem Browsing Toolkit is to enable developers to conceptualize, design, and implement applications that simultaneously cater to multiple displays and/or users sharing a single application session. The toolkit helps developers to support tandem browsing from the ground up with standard web technologies. When the flexibility of web technologies is combined with multi-display interfaces, various use cases emerge. For example, collaborative web applications [1, 2] have potential to enrich conventional browsing by allowing multiple users to take part simultaneously. Another potential use case for tandem browsing is the design of distributed user interfaces for information kiosks and pervasive displays [7, 8].

3.1 A Web Philosophy

The toolkit aims to be seamlessly integrated into the traditional web development process. This means that toolkit is compatible with all web development tools and techniques that developers are familiar with. Subsequently, the developer has to consider at the design phase how tandem browsing will be utilized, on which parts of the application, with what content, and what device requirements [8]. In practice, the developer simply needs to develop applications taking into account that multiple devices may be accessing the same content or parts of the content simultaneously. Following web standards is crucial as for instance requiring a browser plug-in at the client side would make it hard to view the content on various platforms until the plugin would be ubiquitously available for all browsers.

3.2 Device and User Specific Views

Similar to [3, 4], our toolkit targets that content should be configurable per device type and/or user according to the application requirements. Developers should have the option to introduce separate parts and following our earlier work [5], we refer to the individual parts as *virtual screens* as they disconnect browser windows from actual web content. Thus, a designer can divide an application interface into multiple *virtual screens* that can be assigned for different devices and users flexibly. Hence, the detailed design of a tandem browsing application is left to the developer. We argue that this is a flexible approach enabling the development of various types of web applications.

3.3 Orchestrating Navigation

Hyperlinks between individual web pages are a crucial feature of web. In web, the links allow easy interlinking of information and composing complex web applications from individual web pages. On the contrary, pervasive displays benefit from more strict control over what can be shown on the displays and therefore the content should be based on applications rather than free web browsing. For example, an application may require three pages in a predetermined sequence: "welcome", "main task" and "goodbye". Following our earlier work [5], in a tandem browsing application this sequence is enabled with a *finite state machine*. It has a starting page and describes possible transitions between pages. The finite state machine helps in designing a multi-display application as a whole. However, to facilitate flexible application design, the state machines must be dynamically loadable in runtime similarly to accessing a new web site.

3.4 Separating Responsibilities

The above abstractions effectively divide the user interface design process into two parts: (1) designing the overall structure of the application (i.e. the state machine) and the composition of each *multipart page* as *virtual screens*, and (2) designing the actual content for each *virtual screen*. Virtual screens are plain web pages and thus can be easily moved, copied and replaced. Similar to dynamic web pages, the content of individual virtual screens can also change dynamically, which may require synchronization with other virtual screens. The need for synchronization is completely application specific and can vary from mirroring everything (e.g., collaborative form filling), partial synchronization (e.g., total price of individual baskets) to none (e.g., non-dynamic pages). Due to the varying needs, the synchronization is solved externally using a separate communication middleware UbiBroker [6], but developers are also free to use their own synchronization mechanism.

4. IMPLEMENTATION

4.1 Architecture

We have chosen a proxy-based architecture for our toolkit, similar to [1, 2, 4]. However, in our case the proxy server handles only the content synchronization and orchestrated navigation between multiple devices. The actual contents of virtual screens are loaded directly from respective web servers, outside the proxy server. The proxy server is a Java web application running on a Tomcat server and may optionally use a MySQL database for persistent storage of session information if application state has to survive over proxy server restarts.

Figure 1 shows initial steps of a tandem browsing sequence. As a first step each browser makes a HTTP GET request to get a top frame from the proxy server to establish a tandem browsing session. This request contains a shared *session id* as well as potentially client specific *browser* attribute. A session stays alive as long as at least one of the participating clients is connected to the proxy server. The top frames and the proxy server monitor connection between them and if a connection is lost, they attempt to recover it and finally invalidate it if the loss is permanent.

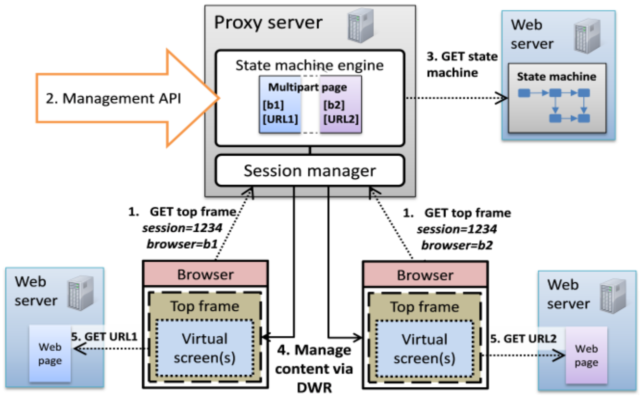


Figure 1. Proxy-based Tandem browsing architecture

In step 2 a state machine is set for the session by sending to the proxy server a *manageSession:setStateMachine* event which contains an URL to the state machine stored on a web server. In step 3, the proxy fetches the state machine, parses it and runs it. The initial state is a multipart page, which defines the placement of virtual screens on the clients. In step 4 the proxy server instructs the top frame on each client to fetch pages *b1→URL1* and *b2→URL2* respectively, based on the XML description of the multipart page and *browser* attributes. This triggers step 5 in which the clients send HTTP GET requests to the web servers to fetch the content of the virtual screens directly outside the proxy. At this stage, state changes within the current state machine as well as new state machines can be triggered via the management API, which renders the content on the clients accordingly.

4.2 State Machines and Multipart Pages

A state machine describes how clients can transition between the various multipart pages of the application. An arbitrary number of multipart pages can be linked together using the finite state machine notation. State machines are suitable for describing predictable applications with fine grained control over which transitions are allowed in which states. The finite state machines are defined using the State Chart XML (SCXML) language, a W3C working draft (http://www.w3.org/TR/scxml/).

We have chosen to define multipart pages declaratively inside the SCXML states in the extensible data model elements following a particular XML schema. The structure of multipart pages is minimal, i.e., inside a root element there can be one or more virtual screen elements. The attributes of the virtual screen elements are: *id, resource, browser, width, height, xPosition, yPosition,* and *zIndex.* They define how the content of a virtual screen is visualized with respect to the full screen real estate of a particular client browser. The *browser* attribute is used in runtime to allocate a virtual screen to the appropriate browser (see the attributes *b1* and *b2* in Figure 1). The attributes *width, height, xPosition,* and *yPosition* use a relative scale from 0…1 and define the region allocated for the virtual screen. The *zIndex* attribute defines the stack order of the virtual screens. The *id* attribute must be unique within the multipart page. The *resource* attribute defines the location of the content of the virtual screen as an URL.

4.3 Tandem Session Synchronization

Synchronization between clients is achieved through a communication between the top frame on each client and the proxy server. The clients sharing the same session id belong to the same tandem session. Proxy server controls the content on clients by manipulating their virtual screens, which are implemented as *iframe* tags. The proxy server communicates with the top frames with the Direct Web Remoting (DWR) library (http://directwebremoting.org). It provides a seamless way to bind server side Java code and client side JavaScript code using AJAX and reverse AJAX. The DWR allows browser sessions to be identified with a flexible attribute-value pair mechanism, which is utilized in managing the tandem browsing session.

Interaction with the proxy server is required to trigger state machine links or to load a new state machine. The management API contains two events with corresponding functions:

- *manageSession* event can trigger two functions for managing client sessions: *setStateMachine* sets the current state machine (allows state machine sequencing); *setParameter* defines data to the shared and individual browser sessions.
- *changeState* event re-directs the browsing session to a new state in the state machine. The event is validated against the state machine in runtime. If the state change is acceptable, the clients involved in the session load a new multipart page.

First state machine for session can be also set as a URL parameter in the request for top frame (step 1 in Figure 1). This allows direct linking to state machines and also nesting state machines. The toolkit supports two alternative implementations of management API: XML over SOAP (for middleware components) and JSON over DWR (for web applications). The interfaces are interchangeable and have identical payload data format.

4.4 Overhead

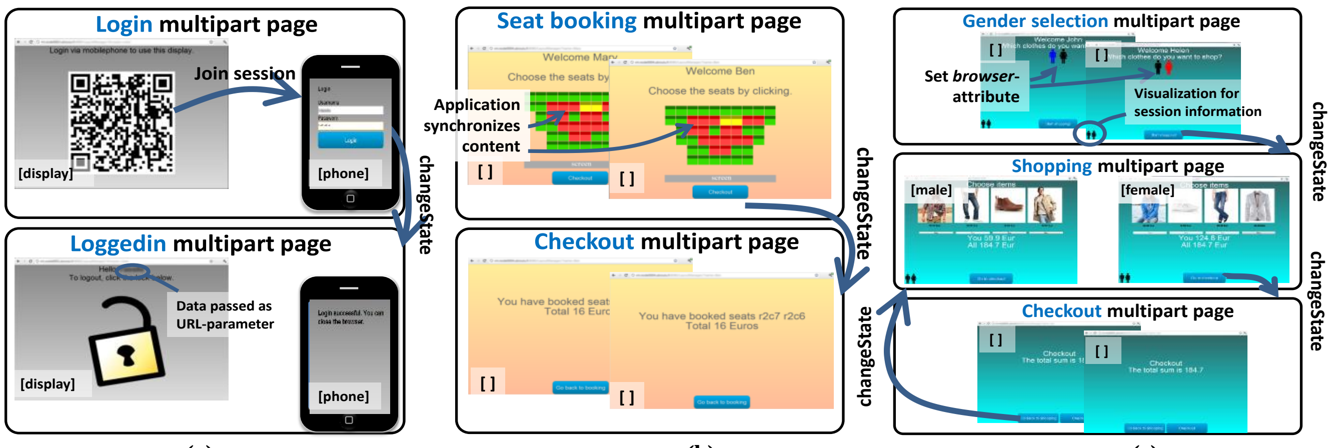
Session initialization involves 12 requests that create 18 kB of additional data transmitted. The management of an ongoing session requires messaging between the top frames and the proxy server. Linking between multipart pages is not direct, i.e., an event needs to be first sent to the proxy server, which will then instruct the top frames in the browsers to request the content of the virtual screens accordingly. Any substantial communication latencies between the clients and the proxy server have an immediate deteriorating effect on the user experience which restricts the placement of the proxy server relative to the clients.

Development overhead consists of two parts: (1) writing the state machines as SCXML files and (2) setting up the proxy server. Similar to other web resource files, developers can copy-paste and modify readymade SCXML files. Also, there are graphical tools for editing, e.g. Scxmlgui (https://code.google.com/p/scxmlgui/). The installation of the proxy server requires an Apache Tomcat server where the proxy server's WAR file is deployed. The proxy server can optionally use persistent storage, in which case the developer needs to deploy a MySQL database.

5. EVALUATION WITH DEVELOPERS

To assess how developers perceive the possibilities of the toolkit and its API, we conducted a workshop with four developers. The participants were researchers or students, and they all had extensive background in web programming which qualified them as potential users of our toolkit.

We first gave a presentation to all participants, discussing tandem browsing in detail and also answered all questions that the participants had about the toolkit's operation. Following the



Login via mobilephone to use this display.

Welcome Ben

Choose the seats by clicking.

You have booked seats r2c7 r2c6
Total 16 Euros

Figure 2. Prototype applications: (a) Tandem authentication; (b) Tandem seat booking; (c) Tandem couples shopping

presentation we had an open-ended group discussion where the participants criticized, praised, and questioned the toolkit's purpose. Along the open-ended discussion we asked a number of pre-defined questions regarding the details about the developer API and implementation. Next, we held a brainstorming session where participants came up with ideas about how to use the toolkit for novel applications.

5.1 Feedback on the Programming Model

In general, a lot of the discussion focused on the separation of the tasks supported by the toolkit and the tasks supported by applications. No participants felt that the toolkit was responsible for too much. Rather, some participants considered that more support from the toolkit could be beneficial, e.g., mechanism for discovering and joining existing tandem browsing sessions.

All participants praised the fact that they can use standard web technologies to implement virtual screens. The web approach was noted as strength as it allows platform independency and seemed a logical choice for implementing multi-display applications. They also considered the rapid prototyping aspect of the web as an advantage, so that pages can be built quickly by copying parts from existing pages.

The programming model of synchronizing navigation between multiple devices was well received. Participants considered it to be most suitable for stateful applications that have multiple clearly distinct "stages". For applications having only a single stage the benefits of the toolkit remained unclear as the toolkit wouldn't synchronize the sessions beyond the point all clients are on the same multipart page. Further, the need for the definition of state machines in every situation was questioned, especially in cases where there is only a single state with one virtual screen, which is shown on each of the client devices. In such case the state machine would be just a wrapper for a single page.

5.2 Brainstorming for Scenarios

The brainstorming session was substantially affected by the participants' prior experience of collaborative websites. For instance, first the participants attempted to draw analogies with familiar concepts such as Google Docs. Then brainstorming focused on a single user using multiple devices and a number of scenarios were identified such as the second screen concept for home. Next the participants focused on scenarios with separate "private" and "public" views across various devices. Finally, brainstorming concluded with multi-user multi-display scenarios, such as a teacher in a classroom and collaborative form-filling.

6. Prototypes

To demonstrate the toolkit's capabilities, the authors implemented functional prototypes of three scenarios identified in the brainstorming session: tandem authentication, tandem seat booking and tandem couples shopping..

6.1 Tandem Authentication

This prototype implements a typical distributed user interface scenario with a situated public display and a mobile phone, as described for example in [11]. In this single-user scenario, the display shows public content and the mobile phone is used to input sensitive information such as passwords.

Figure 2a shows the application flow distributed between the public display and the mobile phone. At the Login multipart page, the fixed display is locked and shows a QR-code containing a URL to join the tandem browsing session. This URL can be opened with a mobile phone having camera and QR-code reader. The URL contains a *session id* -attribute of the fixed display and a *browser*-attribute with value 'phone', which matches the phone specific virtual screen on the Login multipart page. Thus by opening the URL the mobile phone joins the session and gets its specific part of the multipart page. Successful authentication transitions both the mobile phone and the fixed display to a Loggedin multipart page. This multipart page also has different content for the two devices.

6.2 Tandem Seat Booking

This prototype demonstrates a scenario where two users are simultaneously editing shared data. Suppose Mary and Ben want to see a movie together and have to select seats during the booking process. Instead of the other of them making the booking online and communicating with the other via phone, this application allows them to make the reservation collaboratively.

The application flow of the Tandem Seat Booking is shown in Figure 2b. Now, the *Seat booking* multipart page has just a single virtual screen, which shows available seats. Mary and Ben can choose available seats co-operatively by clicking. The synchronization of dynamic content of individual pages is done via the UbiBroker [6]. When Mary and Ben have agreed on the seats, either of them can click the *Checkout* button. This causes a *changeState* event to be triggered on the proxy server. The *Checkout* page contains again only a single virtual screen, which summaries the reservation and outputs the price that are passed using URL parameters.

6.3 Tandem Couples Shopping

This prototype demonstrates a scenario where two users have distinct virtual screens in the application. Imagine Helen and John wanting to shop online together but collecting their personal shopping carts. In traditional web applications they would each have isolated sessions with separate carts. With the Tandem Couples Shopping web application, Helen and John can both simultaneously fill in their individual carts, but at the same time the carts are summed up for a single joint check-out and payment.

Figure 2c shows the application flow of the Tandem Couples Shopping. At the *Gender selection* multipart page Helen and John are both asked to indicate which types of clothes they are interested in (female or male). Their selections are stored in a *browser*-attribute. This is also used as session information by visualizing a person with a corresponding gender at the bottom left corner of the page. This allows Helen and John to see when the other has made his or her choice. Also, this information can be used to monitor, who are actually participating in the tandem browsing session. When changing onto the *Shopping* multipart page, based on the gender the proxy server returns a different virtual screen, men's catalog for John and women's catalog for Helen. As the couple now keeps on adding items to their respective carts, the application synchronizes the total sum of the items between the virtual screens. Again, the dynamic content of different virtual screens is synchronized using the UbiBroker [6]. The *Checkout* multipart page is similar to that of the Tandem Seat Booking prototype, but in this case there is an additional "back" button to return to the previous multipart page, i.e. *Shopping*.

7. DISCUSSION

7.1 Web Based Multi-Display Applications

Our toolkit is aimed at developers and relies on web-based technologies. We argue that these characteristics make our toolkit ideal for developing novel applications for multi-display and/or multi-user scenarios, thanks to the availability of web technologies on most device platforms today. One benefit of web applications is that they avoid the obstacle of installing custom client software on user devices, something that can be challenging, especially in ad-hoc situations. Another benefit is that developers are relieved from the burden of actually delivering the application and making it compatible with a variety of platforms. In this sense, it is crucial that our toolkit does not require any browser plug-ins. Web also simplifies content creation. Web provides a shallow learning curve so that developers gradually take up new and more advanced technologies. This is supported by the fact that working solutions can easily be replicated and adapted by developers. Most resources of web applications are publicly available, and therefore developers can easily benefit from each other's work.

7.2 Separating of Developer Responsibilities

We left out the synchronization of dynamic content of applications between the different parts of a multipart page from our toolkit's core functionalities. We argue that having this functionality in the toolkit is practical only for special cases, i.e., collaborative web browsing, in which case one of the following tools [1, 2, 3, 7] could be used. However, to give the developers a power to design applications for multi-displays that can exploit the individual screens in a flexible way to implement any use case, we argue it is better to provide the synchronization mechanism as a separate service. It is crucial that the application developers can employ their preferred mechanisms that suits best for their use case. We think this is in line with the web philosophy where developers can choose preferred technology for a specific problem in hand and are not restricted to solutions dictated by a particular framework.

Session establishment is another important issue in the division of responsibilities between the toolkit and an application. We have delegated session establishment to applications. The current implementation requires the toolkit to have a unique *session id* for a group of clients belonging to the same session. The value of *session id* can be set freely and it is just a label for forking sessions in the broker. Effectively, the developer has means to define how sessions are created either in design time or dynamically in runtime. For certain cases, it is possible to define sessions statically and direct client browser into a correct session with a predefined *session id*. For instance, if a service is bound to a given physical location and the users use the service in an ordered manner. The Tandem Authentication prototype uses this mechanism as there is only a single session instance running on a specific physical location and therefore the *session id* can be location specific. The two other prototypes, Tandem Seat Booking and Tandem Couples Shopping, exemplify another style of handling sessions. Here applications connect to the proxy spontaneously, i.e., they create a new session upon launch. Here it is expected that the clients have some mechanism to agree on a shared *session id*. This could be a separated meeting room page to which the users connect first and then it re-directs the paired clients to the application. Alternatively, the users could agree on the used *session id* in advance, if the users plan the activity ahead.

A state change will be forced by the proxy server on all clients involved in a particular tandem browsing session. In other words, all clients will move to the target multipart page and its associated virtual screens. Consequently, a user can be surprisingly dragged to a new page due to another user deciding to move on to the next stage in a stateful application. This behavior, however, is completely application specific, so that the decision on which client(s) can initiate a state change needs to be made by the application. For example, an application can implement a specific decision making logic, which requires a certain input from all users before a state change link is activated. Another possibility is to assume that clients have other means for communication, as was the case with the Tandem Seat Booking prototype. There, the actual decision could be made for example vocally and then one user just clicks the link. The Tandem Authentication presented a third way, where the developer defines that one virtual screen is a master with a link that can be clicked. This logic works on scenarios where there is only one user. It could also be used in a coordinator-observer pattern, where one user is for example a teacher controlling a presentation that students follow.

7.3 Security

Security is challenging in ubiquitous environments supporting spontaneous interactions, where mobile devices join and leave ad-hoc interactions with each other [10]. In the current implementation of our toolkit, a client can join any session after it acquires the corresponding *session id*. This may be a wanted behavior in some situations, as in our Tandem Authentication scenario. There, some level of security is enforced by the fact that the user has to physically be in front of the display to access the QR code, as proposed in [9]. However, in some other scenarios leaking *session ids* can present a security risk. Therefore, it is important to provide adequate feedback to users, for instance in form of notifications when a new client joins a tandem browsing session. For example, in the Tandem Couples Shopping prototype such information is given to users by showing a person figure (man or woman) in the bottom left corner of the screen. A third such figure appearing would inform the couple that they no longer have the session just for themselves. Therefore, displaying even minimal session information may improve security.

Generally speaking, establishing a shared secret *session id* is analogous to establishing a shared secret key for subsequent authentication in private key cryptography. There, the parties typically either agree on a key using the Diffie-Hellman key exchange protocol or obtain a key from a trusted KDC (Key Distribution Center). The Tandem Browsing proxy server could also serve as a KDC for its clients.

7.4 Scalability and Fault Tolerance

Our proxy-based architecture is stateful as it manages the state information of the clients. This in turn requires timely communication between the proxy and the clients. Consequently, the scalability of the architecture is limited in terms of geographic scale and the proxy is a potential single point of failure. The placement of the clients, the proxy server and the applications far apart from each other may lead to long delays in synchronization which in turn may result in poor user experience. The scalability over number of clients of the proxy depends on the communication and computing capacity of the server.

These scalability issues can be addressed by replicating the proxy server across sites, thus offering a local and more accessible instance. Ideally, the proxy server should be placed relatively close to the clients as most communication is needed for synchronization between client browsers. As we have described, fetching the actual content of the virtual screens is done outside the proxy server and thus is not affected by the placement and capacity of the proxy server. This reduces the load on our proxy. Nevertheless, we do not expect a single proxy server instance to scale well on a general web browsing setting where thousands of clients are accessing particular resources simultaneously.

If a proxy server becomes unavailable, its clients cannot make new requests via the management API. The toolkit uses following mechanisms to increase fault tolerance. The proxy server has been organized in a way that reloading a top frame will recover the last state of the session. In combination with persistent storage, the sessions can be recovered even after a possible proxy server crash. The call for recovery is made by the top frame in case it notices that a connection to proxy server is lost. This brings the clients to the last visited multipart page, but recovering the virtual screen internal dynamic states is up to the applications themselves.

8. CONCLUSION AND ONGOING WORK

In this paper we presented the tandem browsing toolkit. The toolkit gives developers creative freedom to build online applications that orchestrate and manage multiple devices and users in browsing online content. We demonstrated the capabilities of the toolkit with three functional prototypes, all implemented with open web technologies and without any modifications to user devices. We argue that the web philosophy of the toolkit makes it an attractive option for ubiquitous environments where the ad-hoc nature of interactions suggests that relying on custom software can be challenging.

In our ongoing work we are focusing on two main challenges: the robustness and scalability of the proposed tandem browsing architecture, and more advanced management of tandem browsing sessions. The robustness and scalability of the architecture is being put into test by a number of developers developing tandem browsing applications for our network of interactive public displays deployed at pivotal locations around Oulu [14].

9. AVAILABLE AS OPEN SOURCE

An open source implementation of the Tandem Browsing toolkit, demo videos of the prototypes, and the instructions for using our public Tandem Browsing proxy server for creating tandem browsing apps are available at http://www.tandembrowsing.org.

10. ACKNOWLEDGMENTS

Authors would like to thank the Finnish Funding Agency for Technology and Innovation, the ERDF, the City of Oulu, the Academy of Finland and the UBI consortium for their support.

11. REFERENCES

[1] Atterer, R., Schmidt, A. and Wnuk, M. 2007. A Proxy-Based Infrastructure for Web Application Sharing and Remote Collaboration on Web Pages. In *Proc. Interact'07*, 74-87.

[2] Cabri, G., Leonardi, L. and Zambonelli, F. 1999. Supporting Cooperative WWW Browsing: a Proxy-based Approach. In *Proc. Euromicro Workshop on PDP*, 138-145.

[3] Coles, A., Deliot, E., Melamed, T. and Lansard, K. 2003. A Framework for Coordinated Multi-Modal Browsing with Multiple Clients. In *Proc. WWW'03*, 718-726.

[4] Han, R., Perret, V., Naghshineh, M. 2000. WebSplitter: a unified XML framework for multi-device collaborative Web browsing. In *Proc. CSCW'00*, 221-230.

[5] Heikkinen, T., Lindén, T., Jurmu, M., Kukka, H. and Ojala, T. 2011. Declarative XML-based Layout State Encoding for Managing Screen Real Estate of Interactive Public Displays. In *Proc. MUCS'11*, 82-87.

[6] Heikkinen, T., Luojus, P. and Ojala, T. 2014. UbiBroker: Event-based Communication Architecture for Pervasive Display Networks. In *Proc. PD-Apps'14*, to appear.

[7] Johanson, B., Ponnekanti, S., Sengupta, C. and Fox, A. 2001. Multibrowsing: Moving Web Content across Multiple Displays. In *Proc. UbiComp'01*, 346-353.

[8] Kaviani, N., Lea, R., Fels, S. and Finke, M. 2012. Investigating a Design Space for Multidevice Environments. *International Journal of Human-Computer Interaction.* 28, 11 (2012), 722-729.

[9] Kindberg, T., Bevan, C., O'Neill, E., Mitchell, J., Grimmett, J. and Woodgate, D. 2009. Authenticating ubiquitous services: a study of wireless hotspot access. In *Proc. UbiComp'09*, 115-124.

[10] Kindberg T. and Zhang K. 2003. Secure Spontaneous Device Association. In *Proc. UbiComp'03*, 124-131.

[11] Larsson, A. and Berglund, E. 2004. Programming Ubiquitous Software Applications: Requirements for Distributed User Interface. In *Proc. SEKE'04*, 246-251.

[12] Lindén, T., Heikkinen, T., Ojala, T., Kukka, H. and Jurmu, M. 2010. Web-based framework for spatiotemporal screen real estate management of interactive public displays. In *Proc. WWW'10*, 1277-1280.

[13] Maekawa, T., Hara, T., Nishio, S. 2006. A Collaborative Web Browsing System for Multiple Mobile Users. In *Proc. PerCom'06*, 22-35.

[14] Ojala, T., Kostakos, V., Kukka, H., Heikkinen, T., Lindén, T., Jurmu, M., Hosio, S., Kruger, F. and Zanni, D. 2012. Multipurpose interactive public displays in the wild: Three years later. *Computer.* 45, 5 (May 2012), 42-49.