MoCHA: Augmenting Pervasive Displays through Mobile Devices and Web-based Technologies

Elena Oat, Mario Di Francesco, and Tuomas Aura
Department of Computer Science and Engineering
Aalto University School of Science, Finland
{elena.oat, mario.di.francesco, tuomas.aura}@aalto.fi

***Abstract*—In this article, we propose a web-based system, called MoCHA display, for rich interactions between pervasive displays and users with a mobile device. Our scheme consists of a web-centric architecture, in which both the screen and the mobile device simply need to run a web browser. Our contributions are twofold. First, we develop a distributed architecture based on modern web technologies and cloud computing to provide rich interactions between users equipped with a mobile device and pervasive screens. Second, we devise a mechanism for practical and secure binding of screens and mobile devices based on their proximity. We present a prototype of the system as well as some preliminary results about its usability and performance.**

***Keywords*-Pervasive displays, secure device binding, web-based technologies, WebSocket, HTML5, QR codes.**

I. Introduction

Due to technological advances in the production of displays, recent years have seen a proliferation of high-resolution flat screens of sizes large enough to be employed not only at home, but also in shared or even public spaces. In the latter case, displays are usually connected to the Internet and show content that is dynamic. However, such displays do not allow nearby users to impact on the displayed content in most cases. As a result, they resemble traditional TVs as mere output devices with very limited or even no control on them.

In contrast, public displays have the potential to enable unprecedented applications and services. Indeed, this would require to engage users, for instance, by explicitly allowing them to interact. Unfortunately, most displays are not equipped with input devices. Some screens have a touch input, but they are much more expensive than regular ones. Furthermore, touch interactions are not possible in many scenarios involving public spaces where displays are not directly reachable by people. To overcome such disadvantages and make interaction with displays truly pervasive, smartphones have been advocated as a key aspect, especially in the context of public spaces [1]. To this end, several approaches have been proposed based on different techniques [2–4]. Most of them rely on a specific application running at the mobile device. This is not very convenient as the user has to download first the application in order to make use of the screen. Furthermore, each pervasive display might have its own system and technology, thus requiring the user to install one application for each system.

An alternative to running native mobile applications consists in using a rich web interface. This is nowadays possible due to new technologies and standards such as HTML5, WebRTC and WebSocket. In fact, these solutions allow rich input from the user not only through the touch screen, but also through sensory inputs, as well as audio and video capture. This allows a web application running at the mobile phone to interact with the screen through a regular web browser. As a consequence, it is more immediate for the user to start interacting with the screen, as starting a web browser at a certain address is the only action required to access the system. This approach is also general, making the implementation of pervasive screens device and platform-independent.

The component running at the display could also be a web browser. This is particularly interesting since modern screens – usually marketed as Smart TVs – embed some communication and networking functionality, including a browser. Even when the browser is not directly supported by the display, it is easy to provide the related functionality through commodity hardware such as set-top boxes and media centers. The service can just be provisioned through the Internet, for instance by exploiting a cloud infrastructure.

In a web-centric scheme, both the screens and the mobile devices need to open a certain address in their browsers. While the screen is pre-configured by the application or service provider, the mobile node needs to know which address to access in order to bind to a specific screen. In this context, it is necessary to provide a way for the screen to advertise the corresponding address to the users in a convenient way. Such advertising process should also be secure, so that only users that are in proximity of the screen can access it. Otherwise, a malicious user could obtain the address and take control on the screen, even from a remote location on the Internet.

In this article, we propose a web-based system, called MoCHA display, for rich interactions between pervasive displays and users with a mobile device. Our scheme consists of a web-friendly architecture in which both the screen and the mobile device simply need to open a web page available at a certain address. Our scheme is web-centric and can be exploited by any web-based application. Our contributions are twofold. First, we develop a distributed architecture based on modern web technologies and cloud computing to provide rich interactions between users equipped with a mobile device and pervasive screens. Second, we devise a mechanism for practical and secure binding between screens and mobile devices based on their proximity. We also present a prototype of the system as well as some preliminary results about its usability and performance.

The rest of the article is organized as follows. Section II presents the MoCHA display system with focus on architectural aspects and the supported functionality. Section III discusses a prototype implementation. Section IV compares the proposed solution with the related work. Finally, Section V concludes the article.

II. MoCHA Display

In the following, we will introduce the Mobile Controlled Hybrid Augmented (MoCHA) display system. Before proceeding further, let us summarize the design objectives of the proposed solution. Specifically, the system should:

- leverage web-based standards and technologies;
- provide a mechanism to bind displays and mobile devices based on physical proximity;
- support multiple modes of interaction;
- support multiple services and different roles within a specific service.

A. System architecture

The different components of the proposed system are shown in Figure 1. On one side, there is a *display* (or *screen*). The display shows content that dynamically changes, depending on the requirements of a specific service or application, and also based on user interaction. On the other side, there is a *mobile device*. The mobile device can take control of the display or, at least, influence the content shown by expressing some preference. The display and the mobile devices are assumed to have an Internet connection through which the service can be accessed. Specifically, the interaction between displays and mobile devices is relayed through a *server* which has two major functions. First, it provides the content for the different sides of the service, i.e., for the display and for the mobile device. Second, it enables the communication between the two different components. Note that the server could be either co-located with the screen (if its hardware allows) or deployed in the cloud. For maximum flexibility, including the suitability for scenarios where Network Address Translation (NAT) is employed, we will assume in the following that the server is deployed in the cloud.

Our architecture is web-centric, namely, the service is implemented as a distributed web application. As a consequence, both displays and mobile devices use a web browser to access the service. In order to interconnect a specific display with a certain mobile device, our system supports the secure association scheme that is described next.

B. Display binding

Binding is the process that allows to establish a logical link between a mobile device and a display. As our system is web-centric, binding occurs through a special Uniform Resource Locator (URL) that is advertised by the screen. We refer to this specific URL as *access URL*. The mobile device binds to a display by just accessing its corresponding access URL. The rationale behind this choice is to make the use of the system as fast and intuitive as possible. However, URLs do not provide any means for authentication or access control by themselves. As a consequence, whoever knows the access URL (including malicious users or attackers) could potentially take over the display, even remotely. This would compromise the purpose of the system. Thus, it is important to restrict the use of the system only to the users that are physically close to the display, i.e., that are in its proximity.

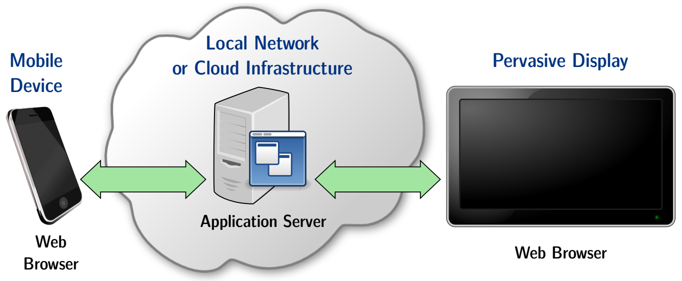


Fig. 1. System architecture

To this end, the access URL changes dynamically so that only users that are in proximity of the display are allowed to successfully complete the binding process (Figure 2). Specifically, the URL embeds a screen identifier and a token that has a limited temporal validity, namely, a Time-based One Time Pad (TOTP) [5]. Once the mobile device opens the access URL, the server compares the token with the expected one related to the specific screen ID. If the two tokens match, then the link between the mobile device and that particular screen is established and a corresponding session is started. Otherwise, an error message is displayed at the mobile device. Note that the validity of the token should be set to a value long enough for the user to open the access URL on the mobile device and reach the remote server. We have verified that tokens with a temporal validity of about five seconds or more allow enough time for the user to open the access URL before it expires.

Our solution is general as it does not depend on a specific technology or solution, as long as it can display content that dynamically changes with time. Indeed, the display can advertise the access URL in different ways. The most straightforward option is to show the access URL directly on the screen. The user would then need to open it in the browser by explicitly typing it on the mobile device. However, having the user to do so is not convenient. Furthermore, it may take considerable time for the user to enter the URL, which needs to be opened it before it expires. A better option is to allow the mobile device to acquire the URL directly. There are several mobile-friendly technologies to this end. One is represented by 2D barcodes, such as Quick Response (QR) codes [6]. QR codes embed the access URL in a visual form that can be captured through a camera, which has several advantages. First, it is practical, as a pervasive display can easily show the code and almost all mobile devices (including most tablet PCs) are equipped with a camera. Second, this approach can be used even when the display is not very close to the user, as long as the QR code is large enough in the captured

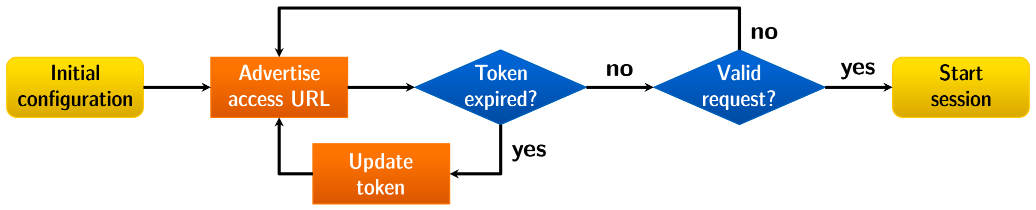


Fig. 2. Workflow of the binding process. Initially, a browser opens a certain service-dependent URL on the pervasive display. The display then starts advertising the access URL, updates the token once it expires, and processes the incoming requests for binding. Requests embedding a valid token are accepted and establish a session between the display and the corresponding mobile device.

image to be reliably decoded. Another option is to use a Near Field Communication (NFC) tag or card-emulation interface attached to the screen. This solution is more intuitive than the previous ones, as the user can just place the phone in contact with the NFC tag to obtain the access URL. However, the NFC technology may not be supported by all mobile device platforms. Furthermore, the display needs to be equipped with an active NFC tag (as the access URL dynamically changes) and needs to be in very close proximity to the user.

C. Service provisioning

Once binding is successful, a session is established and the display is either locked or marked as in use, depending on whether the service requires exclusive access or not. From the user perspective, a session starts with a welcome message that provides basic instructions on how to use the system. Then, the user interface at the mobile device presents a list of services that are configured for the specific display. That is, our system has inherent support for multiple applications and services. The user can select one of the services and start interacting with the display accordingly. The selection of a certain service also triggers the corresponding web application at the screen.

Our system supports multiple roles within each service. Clearly, the display has a special role of output device that changes its content based on the interactions with the user. Besides, a user can be a *controller*, a *guest* or a *peer*. A controller obtains exclusive use of the display. This role is useful, for instance, to display content to an audience. A guest can join an existing session (already established by the controller) to perform application-specific actions which, however, do not directly affect the displayed content. An example is represented by users willing to provide feedback to a speaker during a presentation. Finally, a peer is a user in a group whose members have the same capabilities for interacting with the display. For instance, peers could interact with a public display installed at a transportation hub by expressing their preference on the displayed content. The actual content is not immediately affected by the peers, even though the collective preferences expressed by multiple users have an impact on it.

The proposed system also supports groups of displays that participate in the same distributed web application. For instance, multiple screens can be synchronized and show the same content to the audience. This is particularly useful to extend the reach of the displays, especially in large areas. In order to support groups of screens, the binding process is modified as follows. The displays belonging to a specific group are pre-configured with the same group ID. The latter information is used to calculate the time-based token and to verify its validity. Once the binding is accomplished, a session is established between a user and all the displays belonging to the group that collectively run the distributed web application.

III. Prototype Implementation

In the following, we will provide some implementation details about a prototype of the system, with focus on the protocols and the technologies used.

A. Communication technologies and standards

The communication between the mobile device and the display is realized through the WebSocket protocol [7]. WebSocket is a solution allowing web applications to make use of bidirectional links, similarly to regular network sockets, while providing compatibility with the existing Hypertext Transfer Protocol (HTTP) infrastructure. The major advantage of WebSocket over plain HTTP is the significantly lower overhead in terms of both message headers and connection management. As a consequence, WebSocket enables low-latency communication that is even suitable for real-time web applications.

As regular network sockets, a WebSocket connection only carries the raw data coming from the application. However, WebSocket also supports *subprotocols* that can extend its functionality and make interfacing with applications easier. Among them, the WebSocket Application Messaging Protocol (WAMP) is a subprotocol that offers two asynchronous messaging patterns, namely, Remote Procedure Call (RPC) and publish/subscribe [8]. WAMP allows to easily implement distributed applications, including those related to pervasive displays. Specifically, its RPC pattern enables not only to configure a distributed application but also to manage the different components involved. Furthermore, its publish/subscribe paradigm allows flexible many-to-one communications, which is particularly useful when either displays or users form groups.

In our prototype, we used the open-source implementation of WAMP from the Autobahn project[1]. Specifically, we used the AutobahnJS WAMP client library (written in JavaScript)

[1] `http://autobahn.ws/`

for the components running in the browser at both the display and the mobile device. We also used the the AutobahnPython WebSocket/WAMP framework at the server.

B. User interface and user experience

Both the display and the mobile device show their content through an HTML5 page. While the page at the display is meant to deliver the actual content, the one at the mobile device targets the related user interactions.

In order to provide a rich user experience across different types of devices with varying screen types and resolutions, it is important to adapt the displayed content accordingly. To this end, we have exploited the concept of *responsive design*, that can be easily implemented in webpages through CSS3 media queries and JavaScript libraries [9]. Responsive design is also useful to address the heterogeneity of mobile devices, that could belong to different classes (e.g., tablets or smartphones) and could have different screen sizes and pixel densities. To further improve the experience of mobile users and provide a consistent user interface across multiple platforms, we used the jQuery Mobile framework[2] and additional extensions to mimic the touch events supported by native mobile applications (e.g., swipe and pinch) in web pages.

Besides touch input, our solution supports several other sensors at the mobile phone through the corresponding HTML5 and Document Object Model (DOM) features. For instance, detection of location is supported through the GeoLocation API; audio and video inputs are accessible through the WebRTC `getUserMedia` API; accelerometers readings are available through the Orientation API. Such features enable to turn the smartphone into a sophisticated input device for the pervasive display.

C. Use case: spontaneous presentations

In order to evaluate the feasibility of the proposed approach, we have defined a use case consisting of a spontaneous presentation. The reference scenario is represented by a number of people that are discussing a project in a shared space, in either public or enterprise settings, during a break. Those people happen to be next to a pervasive display and they would like to exploit it to present relevant material about the project in the form of a slideshow.

Our implementation supports the following features.

- One person can request exclusive access to the display. After successful association, such person acts as the *controller*. The controller can upload a presentation or retrieve an existing one hosted by a third-party slideshow service provider. After the slideshow is selected, the screen starts showing it and the controller is provided with an interface that can go to the previous and next slides through both tapping buttons or swipe gestures. The controller can also terminate the presentation to release the pervasive display for other users.
- The rest of the audience can join the presentation as *guests*. Guests can get the content of the presentation on their own device and navigate the slides at their own pace. Once the controller advances to a certain slide, the content displayed at the guests is updated accordingly.

Besides the core presentation service, our implementation also includes other features. One of them allows guests to provide feedback on the presentation in real time. To this end, guests can leave messages on a shared board, for instance, to ask questions related to specific content of the presentation, or just to express some comments. Another feature is represented by interactive polls that can be exploited by the controller to collect the preferences of the audience for further actions.

All features are implemented as separate services and can be selected once binding with a screen (or a group of them) is accomplished. The separation of functionality across multiple services is useful not only from the implementation perspective but also in terms of its inherent flexibility.

1) Software aspects: We based our implementation of the slideshow service on the reveal.js HTML presentation framework[3]. We extended the framework to provide synchronized display of slideshows through WAMP. Specifically, we adopted the publish/subscribe communication paradigm where the controller publishes a topic (i.e., the presentation control channel), which is then subscribed to by both the displays and the guests. As a result, when the controller moves to a different slide, the subscribers update the displayed content accordingly.

As for display binding, we exploited a time-varying QR code to ease interfacing with the existing infrastructure. We implemented a token-based authentication method according to the TOTP specifications [5].

2) Architecture and deployment: The system architecture follows the one depicted in Figure 1. In our implementation, the server component was running as a virtualized instance on the Amazon EC2 infrastructure (specifically, at the US East data center). Besides the software solutions introduced in Section III-A, we used nginx[4] to serve static content (i.e., webpages and media) and python for the application logic.

We have deployed our system in a departmental meeting room that is freely available to both students and staff (Figure 3). The room is equipped with a high-definition TV which was employed as a display. As the TV itself does not have a built-in web browser, we have used a Raspberry Pi board as an inexpensive web-based media center. The board ran Linux and was connected to the Internet with a wired Ethernet connection. The board also opened a pre-configured URL address with a web browser in full-screen mode.

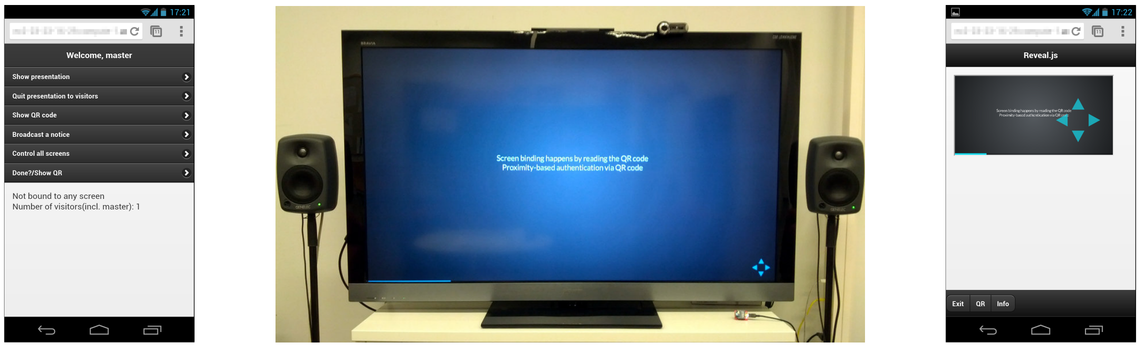
Besides the binding based on QR codes, we also experimented with NFC. To this end, we attached a USB smart-card reader to the board and configured it to act as an active NFC tag. As the HTML5 specifications do not currently support NFC, we wrote a simple program to advertise the access URL.

3) Preliminary evaluation: We have performed a small-scale preliminary evaluation of the usability and performance of the proposed system. Our tests involved graduate stu-

[2] `http://jquerymobile.com/`

[3] `http://revealjs.com/`

[4] `http://nginx.org/en/`


17:21
Welcome, master
Show presentation
Quit presentation to visitors
Show QR code
Broadcast a notice
Control all screens
Done?/Show QR
Not bound to any screen
Number of visitors(incl. master): 1
Screen binding happens by reading the QR code
Proximity-based authentication via QR code
17:22
Reveal.js
Exit
QR
Info

(a) (b) (c)

Fig. 3. (a) Welcome page displayed by the mobile device after successful binding. Note how the system supports different applications. Presentation service: (b) content displayed by the screen and (c) the related webpage at the mobile device.

dents and staff at the Department of Computer Science and Engineering of Aalto University. Generally, the participants have appreciated the functionality provided by the system, with special reference to the presentation service. They have expressed some concern about the media formats supported, which is indeed a limitation of the current system. As for the binding process, scanning a QR code was considered user-friendly. The use of the NFC tag was more suitable for the controller who was anyway physically very close to the screen to reach the tag. In many cases, however, the location (or even the presence) of the tag was not apparent, thus creating some confusion. As for the communication performance, the system was considered responsible enough, even though the cloud data center was located in the US. We have observed round-trip times always below 120 ms, measured in terms of the time elapsed between publishing an event and receiving the corresponding update. This establishes the feasibility of a cloud platform for web-based services for pervasive displays through mobile devices.

IV. Related work

In the following, we compare our system with relevant solutions available in the literature. For clarity, we group the related work in the two categories below.

A. Secure device binding

Secure binding of devices has been deeply investigated in the context of pervasive computing [10]. To this end, most of the proposed solutions focus on user-friendly approaches for context-based authentication or device pairing. The most representative type of context-based device binding is proximity-based authentication, in which devices can exploit shared observations of their surrounding environment in order to establish their physical proximity. For instance, features extracted from the radio channel were used to create a secure connection between devices in [11]. A similar approach based on ambient sound was exploited by [12]. Other solutions, instead, relied on explicit sensory inputs instead of environmental data. For instance, the approach in [13] exploited accelerometer data generated by simultaneous shaking of two mobile devices, whereas the solution in [14] relied on synchronized drawing.

Despite being practical, the above-mentioned methods cannot be used for scenarios involving pervasive displays. In fact, approaches relying on environmental data, such as ambient noise, may not be very reliable in potentially crowded scenarios such as public spaces. Furthermore, many solutions rely on the participating devices to be very close to each other, which might not be the case in urban display networks. Methods exploiting sensory inputs cannot be applied either because displays are not equipped with suitable hardware. Finally, all proposed solutions are not browser-friendly as they require some special support from the application that is not available through web-based technologies and standards. In contrast, we propose a method that only relies on a URL to perform secure device binding and can thus be implemented through several technologies.

Some existing solutions actually provided methods for binding a mobile device to a screen or display. For instance, the Pult web-based media service exploits a four letter code that allows a mobile device to control a remote display [15]. The user has to explicitly enter the code shown on a screen in a mobile device to establish a connection, which is not very practical. In order to improve usability, technologies such as 2D barcodes and Near Field Communication (NFC) can be used. For instance, PresiShare [16] used Quick Response (QR) codes to allow mobile users to share content on a pervasive display. Furthermore, the work in [17] explored an NFC-based interaction technique for large displays. Unfortunately, these solutions did not consider aspects related to security. In contrast, we design a practical and web-friendly solution for secure binding of pervasive displays and mobile devices.

B. Web-based solutions for pervasive displays

Web-based interactions with displays have been considered in the context of pervasive computing for almost twenty years. Among them, the work in [18] proposed a solution for showing web pages on multiple displays by extending the reach of a browser running on a mobile device. The proposed solution is fully based on web standards thus platform-independent and suitable for web-centric content. It also supports a simple access control method allowing to mark the displayed content as either public or private. However, the considered application scenario is very specific and cannot be extended to services different from showing web pages. More recently, PresiShare [16] has been proposed as a web-friendly solution tailored for showing different types of content on pervasive screens through mobile devices. PresiShare is fully web-based, and it exploits the RESTful paradigm for the communication between the different components. Our solution is somewhat similar; however, we do not restrict the interactions with the public display to showing certain file types, but we rather support any web-based service.

There is abundance of frameworks for pervasive computing, especially in the form of middleware. Among solutions specifically targeted to pervasive and public displays, MagicBroker [19] provided a set of abstractions that are exposed as web services through a RESTful API. In contrast, we leverage the WebSocket protocol to achieve low-latency bi-directional communications and HTML5 for rich interaction between mobile devices and public displays. From the systems perspective, [20] proposed an architecture based on virtualization and web technologies for pervasive displays running multiple applications at the same time. Even though the proposed solution explicitly mentions using a browser at the pervasive display, it does not elaborate how mobile devices can interact with the system. Instead, we propose a coherent solution in which both the mobile devices and pervasive displays build on web technologies to provide applications and services.

V. Conclusion

In this article, we have proposed MoCHA display, a web-based system for interacting with pervasive screens through mobile devices. Our solution is based on modern web technologies including WebSocket/HTML5 and implements a mechanism for secure binding. Through a prototype implementation, we have established the feasibility of our approach for a use case represented by spontaneous presentations. We are currently performing a user study to better evaluate the user experience.

Acknowledgment

This work was supported by TEKES as part of the Internet of Things program of DIGILE (the Finnish Strategic Center for Science, Technology and Innovation in the field of ICT and digital business).

References

[1] S. Clinch, "Smartphones and pervasive public displays," *IEEE Pervasive Computing*, vol. 12, no. 1, pp. 92–95, 2013.

[2] N. Pears, D. G. Jackson, and P. Olivier, "Smart phone interaction with registered displays," *IEEE Pervasive Computing*, vol. 8, no. 2, pp. 14–21, April 2009.

[3] S. Boring, M. Jurmu, and A. Butz, "Scroll, tilt or move it: Using mobile phones to continuously control pointers on large public displays," in *Proc. of OZCHI '09*, 2009, pp. 161–168.

[4] J. Y. Lee, M. S. Kim, D. W. Seo, S. M. Lee, and J. S. Kim, "Smart and space-aware interactions using smartphones in a shared space," in *Proceedings of MobileHCI '12*, 2012, pp. 53–58.

[5] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, "TOTP: Time-Based One-Time Password Algorithm," Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 6238, May 2011.

[6] H. Kato and K. T. Tan, "Pervasive 2D barcodes for camera phone applications," *IEEE Pervasive Computing*, vol. 6, no. 4, pp. 76–85, 2007.

[7] I. Fette and A. Melnikov, "The WebSocket Protocol," Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 6455, December 2011.

[8] Tavendo GmbH, "The WebSocket Application Messaging Protocol," `http://wamp.ws/spec/`, retrieved on November 12, 2013.

[9] E. Marcotte, *Responsive web design.* A Book Apart, 2011.

[10] T. Kindberg and K. Zhang, "Secure spontaneous device association," in *Proceedings of UbiComp '03*, 2003, pp. 124–131.

[11] A. Varshavsky, A. Scannell, A. LaMarca, and E. Lara, "Amigo: Proximity-based authentication of mobile devices," in *Proc. of UbiComp '07*, 2007, pp. 253–270.

[12] D. Schürmann and S. Sigg, "Secure communication based on ambient audio," *IEEE Transactions on Mobile Computing*, vol. 12, no. 2, pp. 358–370, 2013.

[13] R. Mayrhofer and H. Gellersen, "Shake well before use: Authentication based on accelerometer data," in *Proceedings of PERVASIVE '07*, 2007, pp. 144–161.

[14] M. Sethi, M. Antikainen, and T. Aura, "Commitment-based device pairing with synchronized drawing," in *Proceedings of PerCom '14*, March 2014.

[15] Pult, "Pult: New way to watch TV," `http://pult.io`, retrieved on November 11, 2013.

[16] M. Geel, D. Huguenin, and M. C. Norrie, "Presishare: opportunistic sharing and presentation of content using public displays and QR codes," in *Proceedings of PerDis '13*, 2013, pp. 103–108.

[17] R. Hardy, E. Rukzio, M. Wagner, and M. Paolucci, "Exploring expressive NFC-based mobile phone interaction with large dynamic displays," in *Proc. of NFC '09*, 2009, pp. 36–41.

[18] B. Johanson, S. Ponnekanti, C. Sengupta, and A. Fox, "Multibrowsing: Moving web content across multiple displays," in *Proc. of UbiComp '01*, 2001, pp. 346–353.

[19] A. Erbad, M. Blackstock, A. Friday, R. Lea, and J. Al-Muhtadi, "Magic broker: A middleware toolkit for interactive public displays," in *Proceedings of PerCom '08*, 2008, pp. 509–514.

[20] T. Lindén, T. Heikkinen, V. Kostakos, D. Ferreira, and T. Ojala, "Towards multi-application public interactive displays," in *Proc. of PerDis '12*, 2012, pp. 9:1–9:5.