Understanding the Use of Web Technologies for Applications in Open Display Networks

Constantin Taivan, Rui José, Bruno Silva
Centro Algoritmi
Universidade do Minho
Guimarães, Portugal
{constantin, rui, bruno.silva}@dsi.uminho.pt

***Abstract* – Open display networks represent a new paradigm for large scale networks of public displays that are open to applications and content from third party sources. Web technologies may be particularly interesting as a technological framework for third-party application development in open display networks because of their portability and widespread use. However, there are also significant challenges involved that result from the specificities of this particular usage domain. In this work, we identify and characterize some of those specificities and analyze their implications for the use of web technologies. This contribution builds on our own experience with the development of multiple web-based applications for public displays and will inform the design of new models for this type of applications.**

Keywords – web technologies, open display networks

I. INTRODUCTION

Public displays can be found in all sorts of urban spaces. They often operate under a content management model in which content is orchestrated at a central location and then distributed to the displays just for presentation. Open display networks represent an alternative model in which large scale networks of public displays are open to applications and content from third party sources [1]. In this model, applications, and not content, are the primary driver for the experience offered by public displays. Empowering multiple third-party developers to publish their applications to be used at any display across multiple administrative domains would be a key enabler for rapid innovation and co-creation of value by a global community [2]. Numerous recent examples, such as the App Store and Google Play have demonstrated the immense potential of opening creativity to a wide range of contributors.

While there are other alternative models, e.g. virtual machines [3] or cloudlets [4], in this study we specifically address the use of web technologies as the technological framework for third-party application development and deployment. Web technologies can be particularly valuable in regard to openness, portability and widespread availability. A vast range of tools already exist and many people already have the competences to create all sorts of web content.

Still, the use of web technologies in this particular domain poses many new challenges. While the ability to present web content from a specific URL is not a challenge in itself and is already an integral part of almost any display system, the overall context of how this content is selected, obtained and adapted to the circumstances of a particular display is something that is not well matched by the prevailing web application models. Moreover, web-based applications face limitations in the access to device specific resources (e.g., RFID, sensors).

In this work, we identify and characterize some of the key specificities of applications for public displays and analyze the implications they might have on the ability of web technologies to serve as the technological background for the creation of this type of application. This contribution builds on our own experience with the development and deployment of multiple web-based applications for public displays and will inform the design of new web-based models for display applications (or display apps).

II. RELATED WORK

Our research builds on the analogy of using web technologies for mobile devices. W3C developed a number of technologies that explicitly address the specificities of mobile devices (e.g. network costs and delays, memory and CPU limitations, input differences, context-aware capabilities): CSS Mobile, SVG Tiny and XHTML for Mobile [5]. A similar process occurs with the use of web technologies in TV sets, also addressed by W3C [6]. With the emergence of IP-based TV devices, a.k.a. connected TVs or Smart TVs, web applications can also be made available on TV sets.

Various display prototypes used Web for their infrastructure and applications [7][8][9][10]. The simple inclusion of an application URL is seen as a regular pattern to provide web content or interactive services to a public display. In particular, Social Networking Services (SNAs) are considered as a dynamic user contributed content source that can be easily leveraged for public displays [11][12].

Very often public display installations are conceived as distributed applications and common design goals include ease of deployment and content creation, maintainability and robustness. A reference example is the display infrastructure in Oulu [13], with 12 interactive displays that support the deployment of web-based applications. Their experiences over a period of three years have shown the many specificities of public displays, which mainly result from the public context of this type of installations. In their work [3], an approach based on virtual machines and web technologies was suggested as an appropriate model for supporting application deployment.

Our recent study on creating web-based display applications by third party developers [14] uncovers that developers face a set of challenges such as visual adaptation, managing of content and fault tolerance support. In order to leverage on developers' web experience, two conditions are required: 1) provide a clear description on specificities of display applications and 2) provide appropriate tools to facilitate the development process.

Erbad et al. [7] have investigated the applicability of web technology in their implementation of MAGIC Broker, a web-based middleware toolkit for the development of interactive large screen display applications that provides common abstractions for such applications. Hartmann et al. [15] built HydraScope – a framework for transforming existing web applications into meta-applications for multi-display environments. These meta-applications execute and synchronize multiple copies of web applications in parallel. The research by Lindén et al. [16], present a web-based framework for spatiotemporal management of screen real estate to enable several independent web applications to be executed in parallel on the same display. Memarovic et al. [17] identified a number of challenges when moving from personalized Web content to personalized content for public displays including user identification, profile location, profile content, content tailoring, model refinement and applications that require personalization. In their vision, public display networks require novel approaches for personalization and existing web personalization solutions cannot be used as they are employed in desktop computing environments. While the Web and its set of enabling technologies are attractive for building displays infrastructures and applications, not much is known about the implications that display applications might have on these technologies.

In this paper, we aim to identify what makes a display app different from its desktop and mobile counterparts. Our goal is to reach a generic understanding about the specificities of display apps that can frame the development of many different types of web-based applications across an unknown and diverse set of multi-application displays.

III. Web-based apps for public displays

In this section, we start by clarifying some of our main assumptions about web-based apps for public displays. For the purpose of this work, we consider a display app to be a web-based application whose primary goal is to render content on a public display. Like any other web application, display apps are based on web technologies and standards, e.g., HTML, JavaScript and CSS. Display web apps run on standard web engines or other types of specially tailored web stacks and they encapsulate both content and the means to render that content on screens. The need for supporting disconnected operation and specially tailored content management policies, led us to assume a rich client model in which the core of the application is running on the display node. Each application will have its own JavaScript code to handle on the display side issues such as obtaining and managing the content items that the application will need, caching and prefetching of content, or dealing with network disconnections.

We also assume that these applications entail a clear separation between content creators and particular displays, reflecting the need to develop applications that may potentially be used anywhere and therefore applications must be developed without any assumptions about their execution contexts. This implies dealing with the potentially strong variations in the resources that may be available across locations. Portability, in the sense of being able to work across multiple display platforms, it is the most obvious requirement, but there is also a need to accommodate other differences in the operational environment, e.g., display sizes or interaction modalities, as well as variations in the associated information space.

While we are not claiming that this particular model would be the only possible model for web-based display apps, this is a perspective that has evolved over the years with our ongoing research in this topic. We have developed and deployed multiple display apps based on these properties. These applications were developed as part of our work in Instant Places system [14], a Web-centric platform for place-based screen media.

The applications we created represent a diverse set of requirements that had a key role in the understanding of the use of web technologies for display applications. We have the following applications: *Presences* app shows people's profile present in the place; *Posters* app shows multimedia posters published by visitors; *Football Pins* app shows content associated with visitor preferences; *Place Stream* app lists recent interaction and place related events; Dropbox app allows place owners to present files from a Dropbox folder; *News* app shows selected news feeds; *Facebook* app shows content from selected Facebook page walls and finally *Polls* app shows public polls.

The applications developed have all been made available across multiple deployments where they have been used by local communities on a continued basis. These testbeds have been pivotal in enabling us to assess a more diverse set of requirements and contextual assumptions.

IV. Display Specificities of web Applications

This section consolidates our findings on the identification of the key issues that web applications need to consider when being repurposed for usage in public displays. The discussion is organized around 4 themes: *content management*, *content addressability*, *visual adaptation* and *integration with the* *execution environment*. For each of these themes, we analyze the specific challenges regarding the ability of web technologies to support display applications requirements.

A. Content Management

In traditional interactive web browsing, content selection is assumed to be under the control of a single user, who may at any moment request a new content resource or be prompted to provide any necessary data, including, if needed, authentication data. In a public display system, content presentation can be mainly autonomously determined by the system itself, which must be able to guarantee that any necessary configurations or content selection options must have been done before the display starts presenting content.

A first specificity associated with content management is the need to avoid idle times when fetching content from servers. Idle times may not always be bad, but on a public display people expect the same smoothness and performance in content presentation that they are used to see in traditional television broadcast and even in other existing display systems. Therefore loading time should never correspond to idle presentation time. If the system stops presenting content while the next content is being loaded, the user experience is completely destroyed.

A second specificity is the need to make any content errors transparent to users. In the traditional web browsing experience, when a content resource cannot be obtained, the result is a message error notifying the user about the problem and possibly giving additional indications on how to proceed. On a public display, content loading errors should never result in error messages being shown, because those people who would see the message might have not requested the content and probably cannot act to solve the problem. This means that applications must be able to catch any such errors before they show up infamously on the screen, and report them through some alternative channels so that appropriate corrective action can be taken. It also means that a fallback strategy must be in place to be activated whenever a content loading error is detected. The application itself may have the ability to detect the problem and present an alternative content that is available.

A final specificity is the need to support partially disconnected operation. In a personal browsing scenario, disconnected operation is not normally very relevant. Either it would be limited to content already seen by the user or the system would have to be able to anticipate the intended content. In public displays, where content is often designed around content loops, cycling through the same content multiple times may even be seen as the expected behavior. The ability to maintain a normal or slightly deprecated operation when disconnected is thus essential.

1) Implications for web technologies: Prefetch and cache are two web mechanisms that may be used to address these content management issues. In public displays, prefetch can be more necessary and also more viable because it is easier to identify the resources that may have to be prefetched. There are fewer potential resources to present and there is an application scheduler that will have at least partial information on what to show in the near future. The ability to prefetch content is thus an essential feature for display web apps. Proper prefetch support may significantly improve the reliability of the system, provide better user experience, save communication costs, and improve the scalability of global applications. Currently, prefetch support is available in Firefox[1] and recently in Internet Explorer 11[2]. The Firefox prefetch mechanism uses the HTML *<link>* tags that instruct the browser to begin fetching a given URL. A site author explicitly defines what resources to be fetched in advance by using a relation type of either *"next"* or *"prefetch"* for the respective *<link>* tag definition. Based on these keywords, the browser will preemptively fetch and cache the respective resource. Standardization of this technique is part of the scope of HTML 5 specification – at present a working draft [18].

Additionally, Chrome and IE 11 browsers employ a distinct mechanism called prerendering. While Chrome has just support for prerendering, thus excluding prefetch support, IE 11 provides both features. At the moment, prerendering is an experimental feature in Chrome browser starting with the 13th release[3]. In Chrome, prerendering is triggered by an element added in HTML that tells the browser to fetch and render an extra page in advance of users actually clicking on it. Prerendering differs from prefetch in the way that a browser instead of just downloading the top-level resource (an HTML page), does all the work required to show the page to the user – without actually showing it until the user clicks. Prerendering mechanism behaves in such a way that the prerendered page is already loaded into a background tab, which is not shown to the user. Only when the user clicks on that page, its content is instantly shown in the current viewing page. Thus, from the user's perspective, the page is loaded much faster than before.

Cache can also be helpful in allowing applications to have local access to recently used resources. The caching properties on the web servers should be optimized to instruct browsers that resources are valid for a long period and should be kept in the cache. However, current web browsers offer limited control over their implicit cache mechanisms, which represents a major challenge to adapt cache behavior to the specificities of display apps, e.g. support for disconnected operation.

On the contrary, application cache is now well supported through the Offline Web Applications [19] technology introduced by HTML5 specification. Some application resources are modified rarely or infrequently, such as images, styles, JavaScript or static HTML. The technology brings the capability

[1] https://developer.mozilla.org/en-US/docs/Link_prefetching_FAQ
[2] http://msdn.microsoft.com/en-us/library/ie/dn265039(v=vs.85).aspx
[3] https://developers.google.com/chrome/whitepapers/prerender

of a web application to be locally cached and still deliver its functionality while there is no internet connection. An Offline Web Application defines its application *manifest* file that specifies every resource that is needed to run locally. The first time the application is accessed it downloads its resources and will always use them, unless the application manifest changes. The manifest also specifies which resources must always use the network to be fetched and also fallback resources (resources to be used if non-cached resources cannot be downloaded). Supported by the most modern browsers, this technique was designed to overcome the limitations of HTTPs browser or client caching mechanisms. While HTML5 application cache mechanism distinguishes by its simplicity, it is also the subject of technical shortcomings when employed in real world scenarios [20]. For instance, a web browser is not aware of the modifications at the server side of cached resources. To trigger an update, the manifest file itself has to be changed. This limitation makes the caching technique not transparent for the developer. Even worse, in the case when the manifest itself is cached, then no update will be performed at all and this can lead to unpredictable behaviors.

B. Content addressability

A key distinction between a normal web app and a display app is that in the latter there is a much stronger need to systematically handle the data exposed by the application. In a normal user-driven browsing scenario, the issue is mainly about links and navigation menus that the user will invoke as needed. When the content is being consumed by a display system, the issue is mainly about exposing and characterizing the set of content items available in the application to allow the display system to integrate that content into the local content schedule. Exposing content as content items or resources, i.e., atomic presentation units, is not mandatory but is crucial not only for automated scheduling purposes, but also because it can become the enabling element for many other key features such as cache management, prefetching, auditing, logging, scheduling and social interactions around content.

While the nature and number of presentation units is highly dependent on the nature of each application and the type of content it generates, application designers should carefully consider how their application content can be organized as presentation units. In some cases, this is not a suitable model and will not be considered. In other cases, there will be many advantages in structuring applications this way in order to take advantage of existing mechanisms for dealing with web content.

1) Implications for web technologies: The use of resources identifiers is an integral part of web technologies. The exposure of web content in the form of multiple individual resources, each with its own identifier (URL) is even one of the essences of the popular resource-oriented architecture - RESTful mode [21]. A resource-oriented architecture (ROA) is a style of software architecture and programming paradigm for designing and developing software in the form of resources with "RESTful" interfaces. These resources are software components, e.g., pieces of code, data structures that can be reused for distinct goals.

Presentation units might play a similar role as the concept of permalink in blog posts. For instance in a blog scenario, with the emergence of permalinks (permanent links) posts can now have a specific URL that remains the same even when they are no longer visible in the blog front-page. This permanence of the links enables those posts to be linked by other sites and provide a reference that supports many other key web functions, such as searching, traffic measure and comments.

C. Visual Adaptation

We call visual adaptation the process of adjusting the content appearance of a website or web app to the browser screen dimensions. For instance, visual adaptation is performed when a desktop web site adapts its text font size to be legible in a smartphone. While this need for visual adaptation is common in desktop and mobile web usage, the adaptability range in public displays can be much more extreme and the role that users can have in assisting the adaptation process is more limited. In public displays, there is much more uncertainty about possible displays sizes and properties. Content may need to be rendered on small displays or small regions of a large display, but it may also have to fill an entire display wall. Additionally, the position of the display in regard to viewers may also face dramatic variations in distance that will severely affect the adequate visualization of content.

1) Implications for web technologies: Responsive Web Design [22] has become a de facto standard practice in web development targeted at the diversity of web devices. In addition to advocating the principle of device independence, Responsive Web Design encapsulates a set of technologies that allow web applications to automatically adapt their content to the display characteristics. For instance, a responsive web site can employ two different menu styles: when the site is viewed on a desktop computer it may have a horizontally arranged navigation bar, but when the same site is viewed on a mobile device, the style of navigation changes to a vertically organized set of links or buttons. Thus, the main idea of responsive web design is to provide users across a wide range of devices, e.g., smartphones, tablets and desktop, a single source of content that can be easily read and explored with a minimum of resizing, panning and scrolling. While responsive web design can be expected to be part of the visualization solutions for web-based applications for public displays, a few more techniques may be needed to deal with the great heterogeneity of potential presentation containers and with additional elements like viewing distance to the display.

When the level of visual adaptation may require more than simple resizing tricks, display applications may offer *alternative* *views* that are expected to be explicitly selected when the respective application is integrated into a presentation container. A view may be embedded with specific options in regard to the ideal displaying size, orientation and viewing distance. In addition to different viewing assumptions, alternative views may also encapsulate particular knowledge about the most appropriate data to be shown, offering different visualizations of the same data, or visualizations that focus different parts of the application data. A horizontal bar at the bottom of the display, for example, is not expected to simply squeeze the content of a full screen into a tight line. A bar of that type is expected to show some key headlines, possibly scrolling. Similarly, a small window designed for pop-up, is only expected to present short notifications. Each application may specify as many alternative views as suitable.

D. Integration with execution environment

An underlying assumption behind the notion that displays will be open to many applications from third-parties is the idea that any particular application is expected to be one of many that may simultaneously be running on a single display and requires sharing the display resources, e.g., screen real estate or interaction features. This means that a display system will employ optimization protocols between applications themselves and between applications and their execution environment. Application developers do not know a priori the conditions in which their apps will be running. Thus, applications could use these protocols to coordinate between themselves to exhibit an integrated behavior, e.g., avoiding contradictory presentation times.

The optimization protocols could also allow apps to have access to local machine resources, e.g. a camera or a Kinect device, or obtain information about the environment, e.g., display ID or presence of people in the vicinity of the display. It may also help to coordinate the content scheduling process, by allowing the container to inform the app about the best moment to start, stop or prefetch content presentation, and also inform the app about the allocated presentation time. Likewise, the app may inform the container about internal events that are relevant for the scheduling process, such as content loaded or interactions received from users, or it may request additional presentation time, request to be removed from presentation or even take the initiative to request presentation when certain events occur.

In order to optimize network resources usage, display applications should report their possible errors to the execution environment, so that it can channel them more efficiently, e.g., to some application quality service that then informs developers. Ideally, developers should have access to libraries and tools for capturing errors and channeling them appropriately.

1) Implications for web technologies: Security restrictions may raise a few issues for these integrations with the execution environment, mainly because of the different usage assumptions between traditional web browsing and display applications.

For security reasons, web browsers impose the restriction of Same-Origin Policy [23], which says that if a document containing a script is downloaded from a certain web site, the script is allowed to access resources only from the same web site but not from other sites. There are however some techniques and workarounds that are usually helpful in circumventing these restrictions, which are becoming easier to deal with in modern web browsers, such as Cross-Origin Resources Sharing (CORS) [24], JSON-P[4], cross-document messaging, i.e., Web Messaging [25], and the use of a proxy to the required external resource. Web Messaging is a HTML5 technology that allows web applications to communicate with embedded web content from external domains. It is a safe messaging system, which does not allow any cross-site scripting attacks. Web Messaging can also be used for communicating configuration data between the application itself and configuration container or display infrastructure.

The execution of a particular instance of an application on a specific web engine may generate local state that may need to be kept between subsequent invocations of the same application on the same browser. However, every time web content is loaded it will not have any information about previous loading events. With client side state, the web content could keep state between subsequent instantiations. For example, a slideshow application may start iterating photos from the point where it stopped the last time. In this regard, HTML5 specification provides a well-known mechanism, which is Local or Web Storage [26]. Web Storage was firstly introduced as an HTML5 feature and now it is a W3C specification by itself. It introduces two mechanisms to store structured data on the client side: *SessionStorage* and *LocalStorage*. The *SessionStorage* mechanism is conceived for scenarios where the user is carrying out a single transaction or multiple transactions in different windows or tabs at the same time. The data can be accessed by any page from the same domain. *LocalStorage* is designed for storage that covers multiple windows, and lasts beyond the current session. Using LocalStorage, web applications become capable to store megabytes of user data, such as user-authored documents or user's mailboxes.

Web storage is an alternative to HTTP cookies storage mechanism [27]. However, cookies do not really handle well these two cases of client side storage. For instance, in the case of session storage the data can leak from one browser tab to another if the same web application is used, e.g., buying two flight tickets in two browser tabs. Moreover, cookies are transmitted with every request, which makes the storage capacity of cookies quite small and inappropriate for storing of large data sets.

[4] http://json-p.org/

V. CONCLUSIONS

The openness and portability of web technologies are key properties when considering the development and usage of third-party applications in open display networks. However, public displays represent a new frontier for web technologies, with novel usage situations and technical requirements. Similarly to what has happened in other domains, e.g. mobile web apps, there is a need for specific techniques that enable display apps to seamlessly integrate the content they generate on the presentation context of public displays.

As part of an urban deployment of public displays, we have created and deployed a number of applications, with different characteristics and requirements. Based on these experiences, we have consolidated our view on the best ways to adapt web technologies usage for the creation of display applications. We highlighted that while the Web has various technological building blocks that can serve our scope, display apps have a number of specificities with important implications on how web technologies can be used in this context. This research is an initial contribution towards a better understanding of those specificities and how they may shape the emergence of new web-based models for display apps.

ACKNOWLEDGMENT

The research leading to these results has received funding from "Fundação para a Ciência e a Tecnologia" under research grant SFRH/BD/75868/2011.

REFERENCES

[1] N. Davies, M. Langheinrich, R. José, and A. Schmidt, "Open Display Networks: A Communications Medium for the 21st Century," IEEE Computer, pp. 58–64, May-2012.

[2] C. Taivan and R. José, "An application framework for open application development and distribution in pervasive display networks," in Proceedings of the 2011th Confederated international conference on On the move to meaningful internet systems, Berlin, Heidelberg, 2011, pp. 21–25.

[3] T. Lindén, T. Heikkinen, V. Kostakos, D. Ferreira, and T. Ojala, "Towards multi-application public interactive displays," in Proceedings of the 2012 International Symposium on Pervasive Displays - PerDis '12, 2012, New York, NY, USA, ACM.

[4] S. Clinch, A. Friday, J. Harkes, M. Satyanarayanan;, and N. Davies, "How close is close enough? Understanding the role of cloudlets in supporting display appropriation by mobile users.," in PERCOM '12 Proceedings of the 2012 Tenth Annual IEEE International Conference on Pervasive Computing and Communications, 2012.

[5] W3C, "Mobile Web." [Online]. Available: http://www.w3.org/ standards/webdesign/mobilweb. [Accessed: 23-Jan-2014].

[6] W3C, "Web and TV Interest Group." [Online]. Available: http://www.w3.org/standards/webofdevices/tv. [Accessed: 23-Jan-2014].

[7] A. Erbad, M. Blackstock, A. Friday, R. Lea, and J. Al-Muhtadi, "MAGIC Broker: A Middleware Toolkit for Interactive Public Displays," in PERCOM '08 Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications, 2008.

[8] N. Memarovic, I. Elhart, and M. Langheinrich, "FunSquare: First Experiences with Autopoiesic Content," in Proceedings of the 10th International Conference on Mobile and Ubiquitous Multimedia - MUM '11, 2011, pp. 175–184.

[9] F. Alt, T. Kubitza, D. Bial, F. Zaidan, M. Ortel, B. Zurmaar, T. Lewen, A. S. Shirazi, and A. Schmidt, "Digifieds: insights into deploying digital public notice areas in the wild," in Proceedings of the 10th International Conference on Mobile and Ubiquitous Multimedia - MUM '11, 2011

[10] M. Geel, D. Huguenin, and M. C. Norrie, "PresiShare: Opportunistic Sharing and Presentation of Content Using Public Displays and QR Codes," in Proceedings of the 2013 International Symposium on Pervasive Displays - PerDis '13, 2013, New York, NY, USA, ACM.

[11] S. Hosio, H. Kukka, M. Jurmu, T. Ojala, and J. Riekki, "Enhancing interactive public displays with social networking services," in Proceedings of the 9h International Conference on Mobile and Ubiquitous Multimedia - MUM '10, 2010, pp. 23:1–23:9.

[12] I. Elhart, "WE-BAT□: Web Based Application Template for Networked Public Display Applications that Show User Contributed Content," in Poster at PerDis 13, 2013.

[13] T. Ojala, V. Kostakos, H. Kukka, T. Heikkinen, T. Linden, M. Jurmu, S. Hosio, F. Kruger, and D. Zanni, "Multipurpose interactive public displays in the wild: Three years later," IEEE Computer, pp. 42–49, May-2012.

[14] C. Taivan, J. M. Andrade, R. José, B. Silva, H. Pinto, and A. N. Ribeiro, "Development Challenges in Web Apps for Public Displays," in 7th International Conference on Ubiquitous Computing & Ambient Intelligence, UCAmI 13, 2013.

[15] B. Hartmann, M. Beaudouin-lafon, and W. E. Mackay, "HydraScope□: Creating Multi-Surface Meta-Applications Through View Synchronization and Input Multiplexing," in Proceedings of the 2013 International Symposium on Pervasive Displays - PerDis '13, 2013, New York, NY, USA, ACM.

[16] T. Lindén, T. Heikkinen, T. Ojala, H. Kukka, and M. Jurmu, "Web-based framework for spatiotemporal screen real estate management of interactive public displays," in Proceedings of the 19th international conference on World wide web - WWW '10, 2010, p. 1277.

[17] N. Memarovic and M. Langheinrich, "Beyond Web 2 . 0□: Challenges in Personalizing for Networked Public Display Environments," in Pervasive Personalisation Workshop at Pervasive 2010.

[18] W3C, "HTML 5.1 Prefetch." [Online]. Available: http://www.w3.org/ TR/2013/WD-html51-20130528/links.html#link-type-prefetch. [Accessed: 23-Jan-2014].

[19] W3C, "HTML5 Offline Web Applications," Working Draft. [Online]. Available: http://www.w3.org/html/wg/drafts/html/master/ browsers.html #offline. [Accessed: 23-Jan-2014].

[20] W3C, "The Future of Offline Web Applications - W3C Workshop." [Online]. Available: http://www.w3.org/2011/web-apps-ws/Report #fixingapplicationcache. [Accessed: 23-Jan-2014].

[21] L. Richardson and S. Ruby, Restful Web Services. O'Reilly Media; First Edition, 2007, p. 448.

[22] E. Marcotte, Responsive Web Design by Ethan Marcotte. A Book Apart, 2011, p. 143.

[23] W3C, "Same Origin Policy." [Online]. Available: http://www.w3.org/ Security/wiki/Same_Origin_Policy. [Accessed: 23-Jan-2014].

[24] W3C, "Cross-Origin Resource Sharing." [Online]. Available: http://www.w3.org/TR/cors/. [Accessed: 23-Jan-2014].

[25] W3C, "HTML5 Web Messaging." [Online]. Available: http://www.w3.org/ TR/webmessaging/. [Accessed: 23-Jan-2014].

[26] W3C, "Web Storage." [Online]. Available: http://dev.w3.org/html5/ webstorage/. [Accessed: 23-Jan-2014].

[27] IETF, "Cookies." [Online]. Available: http://tools.ietf.org/html/rfc6265. [Accessed:23-Jan-2014].