HTML5 Agents – Mobile Agents for the Web

Kari Systä, Tommi Mikkonen and Laura Järvenpää
Department of Pervasive Computing, Tampere University of Technology, BOX 553, 33101 Tampere, Finland

Keywords: HTML5, Mobile Agents.

Abstract: The emergence of HTML5 allows more applications to be run in browsers. Although most of these applications are often network connected, they can also run in off-line mode and especially after deployment they are not necessarily dependent on any server. We argue that the modern Web infrastructure with HTML5 as such can be an agent platform and mobile agents could be developed in similar way as Web applications. For us the agents can also be end-user applications that the user can send to a server so that the state is preserved and the execution can continue. The user can later fetch the agent to the same client device or to another device. In addition to the mobile agent use cases, the concept also allows users to continue their work later on another device or even allows other users to continue execution in their own devices. The paper presents the overall concept and architecture of HTML5 agents, a number of use cases, the proof-of-concept implementation, and a list of example applications.

1 INTRODUCTION

HTML5, the latest version of standards in the HTML family, extends the applicability of the technology towards client-side applications. Traditionally, Web applications have more or less been acting as user interfaces for applications running in a server, but the goal of HTML5 is to allow the development of complete client-side applications. Consequently the emergence of HTML5 allows more applications to be run in browser, and these applications can be deployed over network by using the standard Web technologies. Moreover, although most of these applications are often network connected, they can also run in off-line mode and especially after deployment they are not necessarily dependent on any server. For us the overall goal of HTML5 to support rich applications is important, and in this paper we do not refer to any specific new technology introduced by HTML5.

The capabilities of HTML5 described above, together with other recent developments, such as increasing pervasiveness of the Web and improving performance of JavaScript virtual machines inside browsers, enable the introduction of Web-based application platforms and operating systems – something that can be called *Web operating systems*. In these systems applications can be stored in a cache so that downloading is not necessary at the subsequent invocations of the application. For the user this means that the applications do not differ from using traditional installed applications. Examples of such systems include Cloudberry (Taivalsaari and Systä, 2012), Google Chromium OS (http://www.chromium.org/chromium-os) and Firefox OS of Mozilla (http://www.mozilla.org/en-US/firefoxos). Of these systems, Chromium OS is almost a generic operating system for web-enabled devices, whereas Firefox OS and Cloudberry are systems where all user-visible functionality of a smart phone is implemented as HTML5 applications. The listed systems also prove the above claim that HTML5 technology can be used to build complete and advanced applications which in many ways are indistinguishable from traditional, installable binary applications.

From the technological perspective, however, Web operating systems differ from traditional application platforms, because they build on principles of the Web instead of more traditional concepts associated with operating systems and binary applications. Although applications are often cached, they are not necessarily explicitly "installed" and updates can be delivered automatically and without bothering the end user.

The development of Web operating systems and especially Cloudberry (Taivalsaari and Systä, 2012)

Systä K., Mikkonen T. and Järvenpää L..
HTML5 Agents – Mobile Agents for the Web.
DOI: 10.5220/0004368800370044
In *Proceedings of the 9th International Conference on Web Information Systems and Technologies* (WEBIST-2013), pages 37-44
ISBN: 978-989-8565-54-9

have inspired an idea of *Cloud browser* (Taivalsaari et al., 2013). In Cloud browser the browser session – in essence a set of active "pages" and HTML5 applications, both of which are identified by associated URLs – constitutes the user session. This user session – the URLs of applications and information on the state of the applications – is stored in the Web.

The Web is based on mobile code. Four paradigms of mobile code have been presented in (Carzaniga et al., 1997):

- *Client-Server* where client uses code that is located in another node.
- *Remote Evaluation* where client sends execution instructions, for example SQL queries, to another node.
- *Code On Demand* where code is downloaded to the client for execution. HTML5 applications are widely used examples of code-on-demand.
- *Mobile Agent* where code together with internal state of the application is moved to other node for execution.

The first three paradigms are regularly used in Web applications and (Taivalsaari et al., 2013) presented a concept where the internal state of HTML5 applications were stored in server in the cloud.

This paper builds on the above ideas, where the browser is increasingly acting as the application platform, and where applications can store their internal state in the server for future use. In addition, we propose moving the executable code with the internal state of the application. As a concrete contribution, we propose a system where the applications can also continue their execution while being stored in the server, and the running applications can later be retrieved back to a browser. We consider these applications as *mobile agents* since they comply with commonly used definitions of mobile agents, like the ones presented in (Kotz and Gray, 1999) and (Yu et al., 2006). Although our current applications do not include autonomous migration, the proposed approach supports it at the conceptual level. Our agents are implemented as HTML5 applications and thus we call them *HTML5 agents* in this paper.

We claim that HTML5 technology provides important benefits in implementation and use of mobile agent for two main reasons:

1. HTML5 technologies are widely used and have a strong ecosystem. This means that platforms are widely available and there are de-facto standards and tools.
2. A new class of use cases for mobile agents becomes available since the users can run the same application as a normal application and transfer it to an agent server, and pull it back in another context. This puts the users in control and makes the agent platform more user-oriented.

Many applications can also be implemented with other paradigms presented in (Carzaniga et al., 1997). Especially, some of our use cases could be implemented with Client-Server paradigm so that most of the logic is moved to server and only the user interface is in browser. We believe that our approach is more suitable for cases where:

- local execution gives additional value because of access to local resources, responsiveness requirements or cost or quality issues in network quality, and
- the original source location of application cannot be used as a host for execution, e.g., for commercial or privacy reasons.

In addition, the agents can visit several servers and browser clients during its execution. This enables the use cases that typically require specialized agent platform.

The rest of this paper has been organized as follows. Section 2 describes the proposed architecture, and a number of use cases that can be associated with our approach. Section 3 introduces our proof-of-concept implementation of the framework, and Section 4 discusses some example applications. Section 5 discusses related work, and pinpoints unique features of our approach. Section 6 provides a discussion on the lessons we have learned in the development process and potential ideas for future work. Towards the end of the paper, Section 7 draws some final conclusions.

2 ARCHITECTURE AND USE CASES

In the following, we address two principal elements of our work. First, we discuss how we have realized the system at the level of principal design, and then, we provide some use cases that can be implemented with our system. We place the focus on conceptual level, and do not yet go to concrete technical details of the system, which will be addressed later on in the paper.

2.1 Architecture

In the proposed system, an HTML5 agent can run in two modes:

1. With a user interface inside a client runtime engine, i.e. the browser.
2. In a headless mode, i.e. without a user interface, in an application server that we call agent server in this paper.

The agent can move between these modes and locations when the browser pulls the agent from a server and when the browser pushes the agent back to a server. Furthermore, we support multi-device usage – the browser instance that pulls an executing agent can be different from the browser instance that had originally pushed the agent to the server. Therefore, during its life-cycle the agent may visit several browsers and several agent servers. An example life cycle is presented in Figure 1.

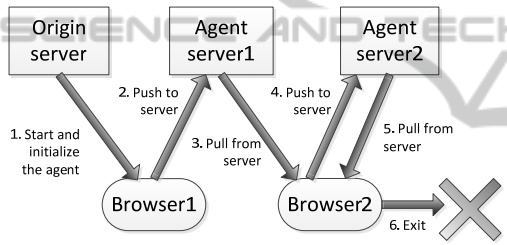

Figure 1: Life-cycle of HTML5 agent.

In Figure 1, agent is started by Browser1, when the agent is downloaded from its origin server (Step 1). In this phase the agent is initialized and the execution begins. Since the agent executes in a browser it has a user interface. In Step 2, the agent is pushed to an agent server. This means that the agent server gets the internal execution state of the agent and the application code (actually a URL to the code). The agent can continue the execution in the server. In Steps 3-5, the agent moves from one environment to other but preserves its internal state and continues execution. Finally, the execution is terminated in Step 6. Note that in life cycle shown in Figure 1, as well as in our experimental implementation, too, the agent always moves between server and client. However, it would be trivial to make the agent to move at least between servers.

It should also be noted that not all applications need to execute in the server, but execution resumes when the agent migrates to (some) browser again. For example, in the media player scenario presented in (Yu et al., 2006) playing should be resumed when agent is in the browser again. However, for some systems, headless mode may provide important functions that are essential for satisfying certain types of requirements. In case of monitoring applications, for instance, certain events might trigger moving the application back to the client so that the user can take appropriate actions. Naturally, this would require presence of some push notification technology.

Internal state of the application is an important part of a mobile agent. The state needs to be serialized, transferred and de-serialized in the new location. It is obvious that without modifying the implementations of the browser the complete state of the application cannot be serialized, and the agent needs to be written so that serialization of the relevant components of the state is possible. Our design provides support for such serialization of the important parts of the state.

In our current implementation a single agent instance moves from host to host, but it would be trivial to change the behaviour so that a new copy is created when needed. For instance when a browser fetches the agent from the agent server, a new copy could be created and the original application could also continue its execution in the agent server. We believe that this can be performed with configuration options included in the framework, but this remains a piece of future work as discussed towards the end of the paper.

Since the agent can run both with and without UI, the architecture has to be designed to separate UI from execution. HTML5 provides a good ground for this separation. Similarly to most HTML5 applications, HTML5 agents are composed of two major parts:

1. Declarative description of the user interface in a form of HTML, CSS and image files.
2. JavaScript-files describing the executable code.

As is common in today's web applications, the HTML-file includes references to Cascading Style Sheets (CSS), images and other resources, and JavaScript files.

When the HTML5 agent is running in the agent server, it runs in the headless mode and the HTML and CSS files are not needed. Only the virtual machine executing JavaScript is needed for the execution of the agent. Naturally, whenever the agent moves to a browser HTML and CSS files need to be available again. Therefore, the agents need to preserve at least URLs of the UI components of the agent. Also the JavaScript code of the agents has to

be written in such a way that it can be executed without presence of the Document Object Model (DOM) tree, the data structure that is used as an internal presentation of a web page inside the browser. (W3C, 2005) We provide a simple browser emulation system that supports running of browser-based applications in a server, but still certain coding rules need to be followed.

The execution model of the application also needs to be suitable for our approach. First of all it needs to be suitable for running in the browser, for instance it should not block the event loop of the browser run-time. On the other hand it needs to execute without UI events delivered by the browser. Furthermore, the agent needs to have safe points in execution in which the internal state can be serialized in consistent state. In practice this means that all the application logic has to be embedded in the event handlers and in addition to handlers for UI events there is a handler for timer events that are generated by our framework.

If the only requirement is that the application and its internal state are stored to an agent server but the application does not need to execute there, there are fewer constraints for the design. The application state must be serialized, but support for the agent execution model – timer-based events – is not needed.

2.2 Use Cases

Our system targets two kinds of use cases: traditional use cases of mobile agents and long-lasting sessions in web applications that are used in multi-device environment.

We fundamentally agree with (Lange and Oshima, 1999) and assume no single killer application for mobile agents, but several applications can benefit from mobile agent technology. From the example application areas presented in (Lange and Oshima, 1999) the following would at least benefit from our solution:

- *e-Commerce*. If the product a user wants is not immediately available, or if the pricing is not satisfactory, the user can set her constraints and then send the e-commerce application to the agent server for execution.
- *Personal Assistance*. The example given in (Lange and Oshima, 1999) ”to schedule a meeting with several other people, a user can send a mobile agent to interact with the agents representing each of the people invited to the meeting. The agents negotiate and establish a meeting time.” would be good use case for us, too.
- *Secure Brokering*. The agent server could be a mutually trusted host that enables collaboration between the agents.
- *Workflow Applications*. The workflow application can be a web application that is executed both in a server and in browsers of several users.

Among web applications all applications that should execute continuously but the user would still like to have a break or just switch to another device are potential use cases of our HTML5 agents. With normal web applications the application state is lost when user switches from one device to another.

Our solution allows users to continue their work in another device. The novel idea presented in this paper is that the execution of the application can also continue while user is not using it through any device. This is beneficial for instance, in the following cases:

- monitoring applications that collect data about events or values of various sensors,
- strategy games that should continue execution of users instructions as robots when user is not present, and
- applications whose execution takes a lot of time or need to access big server-side resources are better executed in the agent server.

Many of these applications can also be implemented as server-side applications that users access with a browser. However, the server-side applications are typically bound to a specific service and configurability for individual users is limited. Some applications would also like to access resources in the client device and thus the execution should take place in the client device. Finally, the ability to move computational agents between clients and servers adds an extra layer of flexibility.

3 PROOF-OF-CONCEPT IMPLEMENTATION

To test and validate our idea we have created an experimental implementation of the agent architecture. The server side of the implementation is based on Node.js technology (Nodejs, n.d.), which is a platform for the development of scalable network applications using JavaScript (Taivalsaari and Systä, 2012). Thanks to Node.js we can execute the same JavaScript code both in server and in browser.

The implementation consists of three main components: 1) *agent framework* that acts as a

superclass for the agent applications, 2) *execution context* in the server-side including an emulation of the browser environment, 3) *serialization format and transfer protocol* for application state and information. More detailed explanations of these components are given in the following.

3.1 The Agent Framework

The execution model is event-based which is normal in browsers and Node.js server. To support execution of possible computing tasks of the agent, a periodic timer calls an application-specific work-function on regular intervals.

As explained earlier, the complete internal state cannot be stored and migrated by the system and also in this experimental system the application need to explicitly define state information that should be serialized and delivered when the agent moves from a location to another. In practice, the agents have *variable list* that contains the state that has been saved and transferred.

The superclass *Agent* is implemented by using the functional inheritance pattern presented in (Crockford, 2008). In this inheritance pattern the super class defines the set of methods as follows:

```
function Agent(src,html) {
  var that = {};
  that.method1 = function(args) {…}
  that.method2 = function(args) {…}
  return that;
}
```

This Agent class is not to be instantiated, but each concrete agent application inherits from Agent as follows:

```
function MyAgent(src,html) {
  var that = Agent(src,html);
  /* new and overridden methods */
}
```

In our design, the class Agent offers the following utility methods:

- `createVar(name,value)` creates a new variable to the variable list, i.e. extends the state of the agent.
- `setVal (name,value)` sets a value of a variable in the state of the agent.
- `getVal(name)` – gets value of a variable in the state of the agent.
- `setWork(function, interval)` – sets the function that is periodically executed with the given interval.
- `setRunInterval(interval)` – changes the interval between executions.
- `start()` and `stop()` – starts/resumes and stops execution of the agent. In practice, the timer is started and stopped.
- `serialize()` – returns a JSON-string that includes all necessary information to preserve the state and continue execution of the agent in a new location. This method is for the framework and is not usually called from application code.

The application-specific sub-class of the Agent can override the following methods:

- Method `getRunningStatus()` – should return a string that the management interface of the agent server context can show.
- Function `continueWork()` – re-initializes the execution when the agent has arrived and de-serialized in a new location and the execution should be resumed. This function initializes the state of the agent by recreating the variables.

In addition, the agent has to provide a function that creates and initializes the agent object.

3.2 Managing Execution Context in the Server Side

As hinted earlier we have implemented a simple HTTP server with Node.js. This server receives description of the agent, i.e., the location of executable JavaScript file(s) and state information, in an HTTP-POST request. Then, the following steps are performed:

- The executable JavaScript file is downloaded from the origin server.
- The required run-time structures are created.
- The function `continueWork()` of the down-loaded agent is called.
- A timer to periodically fire the work-function is initialized.

The application code of web applications typically assumes existence of a DOM tree created from the HTML. For the headless mode we have a minimal *browser emulation* layer for applications that refer to HTML document. The design principle is to provide as minimal emulation as possible and require applications to adapt, too. For example, some of our example applications draw graphics through Canvas API (W3C, 2004). The application code should check if Canvas API is available, in

accordance to the following snippet of code:

```
var ctx = comp.getContext("2d");
if (ctx != null) {
    that.draw(…);
}
```

This would actually be a good coding practice for all web applications since `getContext()` could return null in browsers, too. Our emulation implementation of `getContext()` is simply the following:

```
this.getContext = function () {
   return null;
}
```

In addition, our browser emulation framework includes implementation of standard JavaScript function `getElementById(id)` that always returns a new object that carries the given id and implements our emulation interface. We also have `toString()` that is mainly used for debugging purposes.

In order to use the same API for network configuration, we have installed an implementation of XMLHttpRequest (W3C, 2013) module to our Node.js server.

3.3 Serialization Format and Transfer Protocol

In our design, the browser can receive the agent in two alternative ways:

1. by fetching the HTML-files of the application from the origin server, or
2. by fetching an *agent description* from agent server.

In contrast, the agent server gets the agent by

3. receiving an *agent description* in a HTTP POST message.

In cases 2 and 3 information about the agent and its internal state is encapsulated in an *agent description* that is a JSON document containing the following information:

- *auri*: a URL that points to the JavaScript file of the application. The agent runtime, in the new execution location, needs to download the JavaScript file from this URL.
- *huri*: a URL pointing to HTML file, i.e., UI of the application. The HTML file then refers to required CSS and image files.
- *id*: unique ID of the agent instance.
- *variables*: local state in terms of names and values of local variables.

The utility method `serialize()` of class Agent creates this agent description. When the agent server receives a request to pull an agent (Step 3 in Figure 1) the agent server calls method `serialize()` of running agent and includes the result in the response. When client browser wants to push agent to the agent server, it also calls this method and embeds the result to the HTTP POST request that send the agent to the server.

For optimization reasons, when the application is fetched from agent server, the content of the HTML file is attached to the agent description.

The implementation of our agent system requires the browser to fetch content from several origins. To enable this we have used the Cross-Origin Resource Sharing (CORS) (W3C, 2013) technology to allow use of several origins.

4 EXAMPLE APPLICATIONS

We have experimented and tested the framework with some test applications. The example applications have been used to further develop and validate different aspects of our agent framework.

Monitoring. Our first example application was a simple monitoring application that tracks the CPU load of a server machine. While implementing this application we also made most of our current design decisions. When this agent is run in a browser, the application shows current load level, minimum and maximum values, and latest history graphically. When this application is run in a server, the information is still collected so that a non-interrupted sequence of data can be shown when the agent has migrated back to a browser. This application was tested both on PC browsers and on a smartphone.

Image Analysis. We also wanted to test the framework with an application that contains long-lasting computation and user would like to push the computation to the server after seeing that it has been started correctly. As a concrete example we selected an image analysis application that can compute characteristics of images both in browser and server. With this application we investigated issues with large amounts of data, and the main learning is that the future agents should have pointers in the resources in the Internet instead of carrying the data in the state of the agent.

On-line query applications. We assume that many applications of our HTML5 agents collect information for the user. The motivation could be e-commerce or just to get information when it becomes available. To experiment with this application area we have implemented an agent that lets users to follow an on-line artist community - in our case DeviantArt (Deviant art, n.d) for new content from favourite artists or by selected keywords. With this application we also experimented with a computation that had behave slightly differently in browser and server because the source of RSS feed provides different format and content for different clients.

While the above applications demonstrate the main capabilities of the present implementation, we will continue development of new applications when new and improved capabilities are introduced. Some of these ideas are discussed in Section 6 of this paper.

5 RELATED WORK

As already pointed out, our agents comply with commonly used definitions such as "*Mobile agents are programs that can migrate from host to host in a network, at times and to places of their own choosing. The state of the running program is saved, transported to the new host, and re stored, allowing the program to continue where it left off*" (Kotz and Gray, 1999) or "*Mobile agents are self-contained and identifiable computer programs that can move autonomously within the network and act on behalf of the user or other entities. A mobile agent can execute at a host for a while, halt execution, dispatch itself to another host, and resume execution there*". (Yu et al., 2006)

A survey of mobile agent technologies has been given in (Gupta and Kansal, 2011), but the discussion is limited to traditional mobile agents. In this paper we propose a new approach in which HTML5 - a standard feature of a browser - and the emulation in application server constitute the agent platform.

Benefits and application areas of mobile agents have been discussed in (Lange and Oshima, 1999). From the motivations presented in (Lange and Oshima, 1999), at least reduction of network load and latency, asynchronous and anonymous execution are valid for HTML5 agents, too. Also many of the presented application areas are common to HTML5 agents.

An agent platform hosted in browsers has been presented in (Feldman, 2007). This work has common targets with ours; it allows agents to be executed both in server and browser. In (Feldman, 2007) mobile agent platform is based on concepts of *Pneuna* that is relatively close to our agent description and *Soma* that is the execution environment. In the approach proposed in (Feldman, 2007) Soma hides the differences of browser and server environment and creates a completely new application platform for mobile agents. In our approach standard and well-known HTML5 is the agent/application platform. In addition the approach presented in (Feldman, 2007) has not been designed for pushing agents to agent server when user or browser is not active or when the agent should find a new browser to run on.

The basic idea presented in (Yu et al., 2006) is somewhat similar to our approach, but they do not use HTML5 as the application platform. For example, their media player example would fit nicely to our approach, too, especially if HTML5 media API would be used to implement the player. A particularly interesting aspect in (Yu et al., 2006) is self-adaptation and context awareness – in practice different UI to different devices. It would be interesting to implement similar behaviour using web technologies.

The relation between trends in the Internet and in mobile agents has also been discussed in (Kotz and Gray, 1999). The paper discusses these trends and forecasts that "within a few years, nearly all major Internet sites will be capable of hosting and willing to host some form of mobile code or mobile agents." This paper has also interesting discussions about technical and non-technical hurdles of mobile agents. From the presented technical hurdles using HTML5 overcomes "standardization and portability" and to limited extent also "security". From the non-technical hurdles our approach solves "Getting ahead of the evolutionary path" because we use the de-facto HTML5 technology, and "Revenue and image" has effectively been eliminated by the evolution of the Internet and its business models.

6 FUTURE WORK

We are still in initial phase of our work and have only tested our approach with a limited number of applications. Although we have proven that the approach works with these applications, we anticipate a need to improve the architecture so that development of new agents becomes easier. We need to develop new applications and collect

feedback from other developers. For example, our browser emulation interface for the server side is still incomplete, and it has only been tested with very a limited set of test applications. Therefore new features must and have been added to the system as new applications are implemented.

Context awareness, as in (Yu et al., 2006) should also be added, and we should work more with different types of devices. In particular context awareness should be complemented with mechanisms for self-adaptation, which would provide extra flexibility in some use cases.

Security is an obvious concern for all dynamically moving code. In the current system we rely on standard security mechanisms of HTML5 applications in browser. However, if the agents need to execute sensitive tasks in the server or agents of several users should collaborate in the server, we need to develop additional security mechanisms that are missing from today's Web.

In our current example a single instance of an agent migrates between hosts. Extensions to multiplying agents should be experimented with and we should develop configuration techniques for when to move applications and when to multiply them.

Finally, exploring agents that are able to autonomously extend their behaviour in accordance to the needs of the application is one of the directions we have been considering. Building on the immense flexibility of the Web and web applications, this would be a step towards mashware where components offered as a service form the basis for constructing applications (Mikkonen and Salminen, 2012).

7 CONCLUSIONS

Recent development of web technologies is rapidly gearing the Web towards a role where it offers more and more facilities that have been commonly associated with traditional operating systems and binary applications.

In this paper, we have shown that HTML5 technology can be used to implement mobile agents and that use of agent approach can improve the user experience especially in multi-device scenario. In addition, we introduced a proof-of-concept implementation that is able to run simple applications. While as future work, we list a number of ideas that will improve the capabilities of the system; we believe that the present implementation validates the feasibility of the fundamental design.

REFERENCES

Carzaniga, A., Picco, G., P., Vigna, G., 1997. Designing distributed applications with mobile code paradigms. In Proceeding of the *19th international conference on Software engineering (ICSE'97)*, May 17-23, 1997, Boston, Massachusetts, USA pp 22-32.

Crockford, D., 2008. *JavaScript: the Good Parts*, O'Reilly Media, Inc. May 8, 2008.

DeviantArt n.d., home page. http://www.deviantart.com/, Last viewed 3.2.2013.

Feldman, M., 2007. An approach for using the Web as a Mobile Agent infrastructure, In *proceedings of the International Multiconference on Computer Science and Information Technology*, pp. 39 – 45, 2007.

Gupta, R., Kansal, G., 2011. A Survey on Comparative Study of Mobile Agent Platforms. In *International Journal of Engineering Science and Technology* (IJEST), 3(3): 1943 – 1948.

Kotz K., and Gray, R., S., 1999. Mobile Agents and the Future of the Internet, In *SIGOPS Oper. Syst. Rev.*, 33(3):7–13, pp 7-13.

Lange, D., B., Oshima, M., 1999. Seven good reasons for mobile agents, In *Communications of the ACM*, Volume 42 Issue 3, March 1999, pp 88 – 89.

Mikkonen, T., Salminen, A., 2012. Implementing Mobile Mashware Architecture: Downloadable Components as On-Demand Services. *Procedia Computer Science*. 10, 2012, pp 553-560.

Nodejs, n.d., Web page for document and download of nodejs technology, http://nodejs.org/. Last viewed 03.02.2013.

Taivalsaari, A., Mikkonen, T., Systä, K., 2013. Cloud Browser: Enhancing the Web Browser with Cloud Sessions and Downloadable User Interface", To appear in the *Proceedings of GPC 2013*, Soul Korea, 9-11.5.2013, will be published as LNCS (Lecture Notes in Computer Science).

Taivalsaari, A., Systä, K., 2012. Cloudberry: HTML5 Cloud Phone Platform for Mobile Devices. In *IEEE Software*, July/August 2012, pp.30-35.

W3C, 2004. The Canvas 2D API 1.0 Specification. W3C Editor's Draft. http://dev.w3.org/2006/canvas-api/canvas-2d-api.html. Last viewed 04.02.2013.

W3C, 2005. Document Object Model (DOM). http://www.w3.org/DOM/. Last viewed 04.02.2013.

W3C, 2013. Cross-Origin Resource Sharing. W3C Candidate Recommendation 29 January 2013. http://www.w3.org/TR/cors/. Last viewed 03.02.2013.

Yu, P. Cao, J., Wen W., Lu, J., 2006. Mobile Agent Enabled Application Mobility for Pervasive Computing, In *Lecture Notes in Computer Science* Volume 4159, 2006, pp 648-657.