The Super-Browser: A new Paradigm for Web Applications

Mark Wallis, Frans Henskens, Michael Hannaford
Distributed Computing Research Group
University of Newcastle
Newcastle, Australia
Email: mark.wallis@uon.edu.au, frans.henskens@newcastle.edu.au, michael.hannaford@newcastle.edu.au

***Abstract*—The modern web browser performs a multitude of tasks, which were never contemplated in its original design. This work investigates the roles and responsibilities of the various components that comprise the web browser, and a traditional web application. We review the web browser's role in the greater architecture and propose that a "Super Browser" concept need not mean greater responsibilities for the web browser application. Instead, the solution described in this work introduces a distributed approach that is capable of executing applications composed of distinct components. The paper presents an implementation of this concept, and a comparison of this approach with a traditional web application framework.**

***Keywords*-web browser; cloud; personal data; architecture.**

I. Introduction

Web Browsers were traditionally designed to render static content sourced from remote web servers. Textual content and page format were provided as static HTML and selected multimedia types were then rendered via subsequent requests to the server. As the World Wide Web has evolved, so have the web browsers. Web Browsers today support features such as dynamic content rendering and the execution of pluggable code modules. These features have been gradually added as requirements have arisen, and were never part of the original specification of tasks a web browser was required to perform. As such, the stability and security of web browsers has been brought into question [1]. The latest attempts to address these issues have seen introduction of further features such as HTML5 [2] and execution sandboxing [3]. Simultaneously, web applications have also evolved. Various software development models now exist for the development of web applications and the increased use of Cloud Computing [4], [5] has introduced new concepts and new challenges [6].

The system presented in this work introduces a new way of designing and developing web applications. The role of the web browser is reduced to its originally intended role, that of a content renderer. Web Applications are presented as a set of inter-related components. Components may execute anywhere within the user's environment, and may be specifically tasked with such roles as content generation, data storage, and background processing. A work-in-progress implementation of this design is presented and functionally compared to existing solutions. Metrics are presented showing that this solution has performance that is comparable to existing designs. Finally, we present a summary of future work in this area.

II. Problem Description

There is no doubt that over the past few years, web browsers have had more than their share of performance and security related issues [1]. As the user interface requirements of web applications evolved, the trend has been to push more and more features into the web browser. An alternate design approach would be to revert to traditional thick-applications hosted at the client (an appropriate way, perhaps, of describing the current 'fattening' browsers?), with the client application using network messages to interact with remote databases. It seems clear, however, that web applications do provide many advantages over client-hosted thick applications, for example, ease of deployment and centralised management. During our review of the current technologies it became apparent that each design methodology had its own positives and negatives:

1) The web application approach provides developers with positives in the areas of ease of deployment, centralised management, and standards-based development. On the negative side, web browsers are not consistent in the implementation of the 'standards', so developers are required to consider each specific web browser, making sure that the end product handles specific quirks of the browser implementations and versions. Additionally, the implementation of web applications with fully dynamic user interfaces including high-quality video and user interaction are often problematic, relying on closed-sourced web browser plugins such as Adobe Flash [7] and Microsoft Silverlight [8]. On the positive side, web application technology makes it easier to develop a program that can execute across disparate processor/operating system combinations without having to compile specifically for each environment.
2) Thick applications provide a solid means of implementing tasks (that, incidentally, have proven to require ever increasing client processing capabilities),

but are limited in their utilisation of distributed technology. For example, the distributed deployment scenario in which each user has their own locally installed instance of each application causes problems when it comes to patching, and deploying software updates. There are also limitations when it comes to developing an application that can execute on multiple different platforms, with compilation of a specific binary required for each platform. While languages such as Java have addressed this issue through the use of common byte-code, it locks the developer into only being able to support a single language.

3) The ”application store” concept is a newly emerging architecture that attempts to blend the ease of deployment of web applications with the stability and feature set of thick applications. Application stores provide distribution, and in some cases, patching functionality without limiting execution to within a restricted sandbox, such as the web browser.

It has become clear during this review that the ability to blend the above approaches would be beneficial to both the software developer and the end user. The design presented in the remainder of this paper provides a component-based approach to generation of an environment that takes the best features from the above architectures.

III. System Design

By using a component-based system architecture we can address many of the concerns raised in the preceding section. The design of this system can be viewed from four perspectives: component execution, component communication, user interface, and data storage.

A. Component Execution

A component-based architecture allows a developer to design and build an application using multiple, distinct, independently executing components. With the support of an appropriate runtime environment, these components can be implemented in different programming languages, and can execute in a distributed fashion with components being hosted, perhaps, by different hardware/operating system platforms. This architecture builds on that used for existing web service [9], [10] approaches, in which a web application makes use of a set of services to perform a particular task. The system we describe in this paper extends this concept to present the entire web application as an orchestrated set of components.

Components may be specified as being of the following kinds:

- Execution components present only an interface, which may be called (executed) by other components. They are generally used for processing tasks (such as encryption), and background service tasks (such as external notifications). These components execute within a runtime environment and may be physically deployed across a range of locations such as on a user’s local machine, on a specific server, or in a Cloud Computing environment.
- User interface components integrate with a web browser to generate an HTML5 [2], [11] compatible interface to the user. This provides realisation of the ”super browser” concept without the overhead (and browser ’fattening’) of code execution within the web browser itself. User interface components execute in a runtime environment that is process-separate to the web browser process. Communication between the web browser and the user interface component is performed through a strict interface, implemented in our pilot system as a set of JavaScript functions within the web browser and call-back functions in the user interface component. Security is ensured through process separation and monitors, while stability is addressed by ensuring all calls between the user interface component and the HTML realisation of the interface in the web browser are executed through well defined functions, as opposed to Domain Object Model (DOM) [12] manipulation.
- Data Storage components are similar to execution components, in that they present a callable interface, but in addition they are backed by persistent storage. The interface exposed by these components is tightly coupled to the data object(s) being persisted. Data storage components implement specific data objects, but they may share a common persistence model between them, such as a relational or NoSQL database.
- Finally, bootstrapping components provide an entry point for application orchestration. They are responsible for defining which other components are required to execute an application, and for requesting instantiation of those components. Bootstrapping components are registered in the ”application directory”, so that users can view which applications are being presented by the system.

A specific application many consist of multiple instances of each of the above kinds of component. Each component may execute within a separate execution environment, but logical groupings will be evident in most designs. For example, user interface components would typically execute on the same host as the web browser. Data storage components would often be grouped together, and execute on the same machine that hosts the related data store. It is the responsibility of the component runtime environment to select the appropriate location of execution for each component.

Figure 1 depicts the high-level system design. Component 1 (C1) is a data storage component. C2 is a user interface component and C3/C4 are execution components. The bottom half of the figure depicts a deployment scenario of these components running in a distributed fashion across various environments, including a mobile device, a server farm and a Cloud Computing environment.

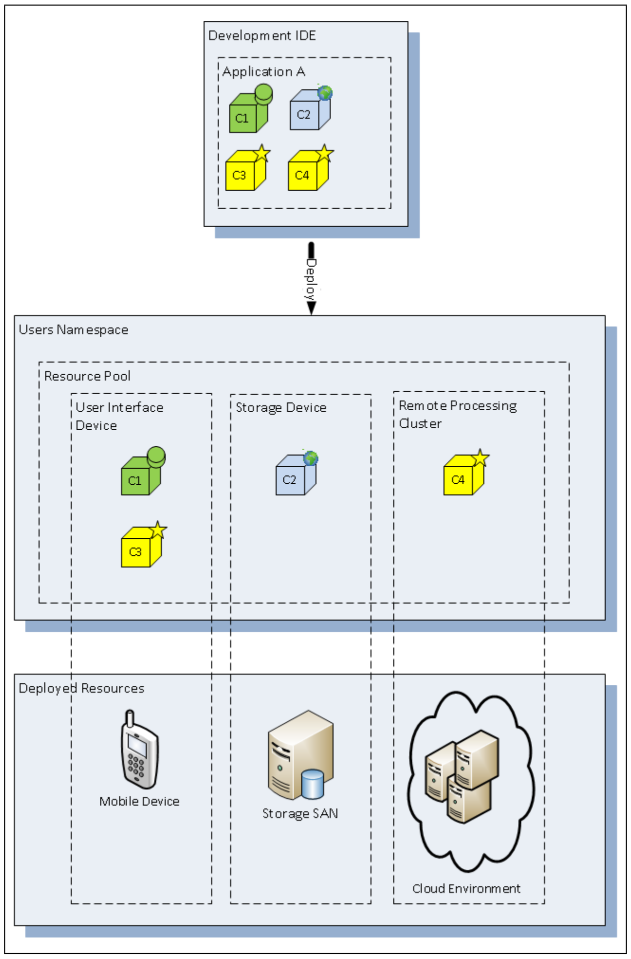

Figure 1. Distributed Component Model

B. Component Communication

Once components are instantiated across various environments, they need the ability to communicate in some fashion. Existing SOA-style architectures rely on each component establishing point-to-point channels for message passing, which provides the means for inter-component communication. While effective in dynamically orchestrated environments, the use of point-to-point channels does not typically scale well. Solutions such as Enterprise-Service-Bus architectures [13] look to address this problem by providing a common communication backbone for inter-component communication, but these buses also do not scale to Internet-wide component execution, as they typically rely on a single (non-global) bus instance. Any inter-bus communication is generally statically configured.

The bus architecture implemented in our design provides a massively distributed channel, with which multiple users can pass messages between components via the same bus, while retaining security and privacy through use of namespace techniques. Each machine that executes/hosts a component has its own local instance of the communication bus. Each bus instance is able to locate and pass messages to other buses using a logically centralised repository of end points. Each user in the environment has their own local endpoint sub-repository, which provides a way of locating specific instances of components.

The communication bus takes care of all inter-component message passing. Components are only required to correctly address and pass messages back up to the hypervisor provided by the runtime environment. Each component is only able to address components within its own application namespace, or components that have been marked as public in the component directory. The communication backbone takes care of resolving these requests to specific instances and locating the specific bus that has local addressing to the addressed component.

C. User Interface

Dynamic user interfaces in traditional web applications rely on Javascript code executing within the web browser runtime, which is capable of directly accessing and modifying the interface via the DOM. Each web browser's implementation of the DOM is known to have its own specific variations from the official specification, and as such this method of building a user interface is common seen as unstable [14].

The user interface component in our proposed design separates code execution from the web browser itself. This separation limits the web browser to simply providing a basic shim that can alter specific DOM elements. Full DOM access is not provided to the executable code, limiting the scope of potential issues caused by non-standard DOMs. The interaction between user interface components and the HTML realisation of the interface within the web browser is implemented via a strictly defined interaction interface. The actual rendering of the HTML5 content is performed by pre-existing rendering engines. This approach removes the need for complicated plugin and closed-sourced components, which can affect the stability of the web browser process.

D. Data Storage

Data Storage in the current generation of web applications is the responsibility of the web application itself. This introduces problems such as data duplication, data freshness, and data ownership [15]. The design for data persistence in our new system is tied closely to that developed in our previous research [16]. Accordingly this new system pushes the responsibility for data storage on to the end user. Data

Storage components will exist for each data object within an application, but the actual persistence framework will be managed by the end users, rather than by web service providers. This ensures that applications are provided a stable interface to data objects, without needing to take actual responsibility for the storage mechanism itself. Data storage components will often execute on servers, or in cloud environments, as examples of locations from which end users are able to procure storage services as required.

Figure 2 shows at a high level how the distributed data storage system works in a Web 2.0 scenario. This proven [16] approach has the data owner publishing their data to the DSS - Data Storage Service. Web Applications subscribe to this information using a publish/subscribe algorithm. Users of the Web Application access both the application itself (for HTML/framework/etc) and the DSS (for direct access to stored data). The user access to the DSS occurs using a SAML-based secure handoff technique and is key in providing a solution that performs in a way that is comparable to existing technologies.

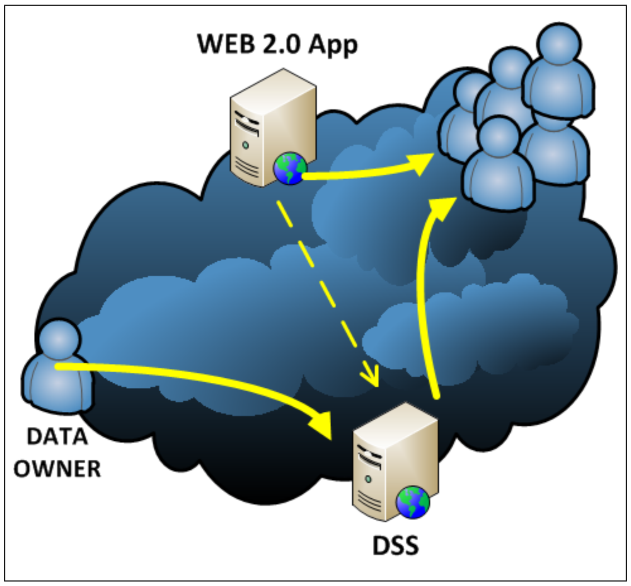

Figure 2. DSS

IV. Implementation

The authors have completed a pilot implementation of the system architecture presented in this paper. The initial deployment provides Java language support for all component types, though the communication channel is not limited to communicating only with components executing within a Java Virtual Machine. Applications can be orchestrated from the bootstrap component, and the components are instantiated in a distributed manner. Runtime support exists for the following environments:

- Microsoft Windows.
- Ubuntu Linux.
- Amazon EC2 Cloud.

User interface support is currently implemented using Gecko, the rendering engine from Mozilla Firefox [17]. The interaction between user interface components and the HTML realisation of the interface is currently implemented through the use of a TCP link with a Firefox web browser plugin. Ultimately, implementation of the rendering engine will be completely split from the traditional web browser process and wrapped in a thinner shim, thus removing much of the unnecessary plugin and sandboxing features that are made redundant by this solution. The component execution environment is implemented within a Java Application Server (Glassfish [18]) and the component communication is realised through the user of a customised Enterprise Service Bus (Glassfish ESB).

V. Evaluation

A functional evaluation has been performed of this solution in comparison to Web 2.0 technologies. Specific benefits have been identified in the following areas.

A. Application Stability

Current web technologies force the developer to implement dynamic user interface actions within complex Javascript or closed-source web browser plugins. The proposed system provides a complete runtime environment (which had historically been limited to thick applications) to the user interface component of the application. Interaction is tightly controlled via callbacks and stub functions. No direct DOM access is provided, which enforces a strict implementation strategy and good programming techniques on the developer. It also reduces overall bloat of the web browser process, and brings stability of the solution through distribution of execution. Each component implements the concept of information hiding [19] which is a proven benefit of component-based software engineering. The internal structure of the browser is hidden away from component code, while in comparison, malicious or unstable Javascript has complete access to the user interface implemented through the DOM.

B. Data Storage

The features provided by the distributed data storage model ensure that users are responsible for their own data storage. Web applications still retain full access to the data they require for execution, but this access is facilitated by remote calls to data storage components, as opposed to direct access to large silos of local storage. The issues of data freshness, data duplication, and data ownership are therefore well addressed by this model, while they remain a key issue with traditional web applications [20].

C. Resource utilisation

A key issue with the increased uptake of web applications is that local resources are becoming highly under-utilised. Local CPUs and data storage are only used to facilitate execution of the web browser process, while the majority of the work is performed by the server(s) hosting the web application. In comparison, the presented system performs execution in a way that allows sharing of load between any resources that a user has rights to use. A specific application may have components executing on the local machine, on a server, or in the cloud, all at the same time, in a distributed and transparent manner.

D. Transaction Support

Transaction managers in current web applications are limited to visibility of tasks executing within the web application itself. For example, if a user closes their web browser mid-transaction then often that transaction is left hanging to time-out. By moving the orchestration of the user interface out of the web browser into a specific user interface component, it becomes possible to track tasks within a transaction all the way from the user interface to the data storage system. This allows software developers to create a transaction that may involve data persistence tasks, calculation tasks, and user interaction tasks, all in the one atomic action and not wholly dependent on continued execution of the interface.

E. Cloud Computing

The proposed design also promotes extended use of Cloud Computing concepts. Currently, for a web application to execute within the cloud it is commonly seen that the complete application must be wholly encompassed in the one cloud environment. Any inter-cloud communication is restricted to application-level interactions such as web service calls and message passing. Code executing in one cloud must specifically be aware that access to code in another cloud requires a manual call over the network. The design presented in this paper abstracts component communication in such a way that building of applications out of inter-cloud components is greatly simplified. Each participating Cloud instance houses its own runtime environment with its own component communication bus. The user's namespace tracks the location of each instantiated component and abstracts the inter-cloud communication back to inter-bus communication.

F. Application Access

This design provides benefits in the area of application access and startup when compared to existing technologies. The bootstrap process is initiated when a user attempts to access an application built according to this component-based design. The bootstrapping ensures that only the initially required components are dynamically downloaded, instantiated and executed. Existing technologies such as Java Web Start [21], in comparison, must automatically download the component application bundle before execution can begin.

VI. METRICS

During the development of this system we regularly compared with current techniques to ensure that existing levels of performance and stability were maintained by our new architecture. Initial metrics have been collected on the following data points:

- Total bytes transferred per web transaction.
- Total round-trip time per web transaction.

These metrics were recorded using a user login event as an example of a typical transaction. The user login event comprised the following high-level tasks:

1) Render static login page to user.
2) Accept user input and perform basic data validation.
3) Encrypt provided user credentials.
4) Pass credentials from user interface to authentication component.
5) Decrypt provided user credentials.
6) Access a persistent data store of user credentials.
7) Authenticate the user and generate a token.
8) Pass the token back through the authentication component to the user interface.
9) Report to the user if authentication was successful.

In the traditional approach this design would be implemented using a web browser, a web application, and a relational database. In comparison, according to our new architecture the following components were generated:

- User interface in HTML5 rendered by the web browser.
- A user interface component executing on the user's local machine (that also hosts the web browser).
- An authentication component executing on a local server.
- An encryption component executing on a local server.
- A data access component for the User object executing in Amazon EC2.
- A relational database also executing within Amazon EC2.

Figure 3 compares the respective total user round-trip time experience for the traditional and proposed implementation architectures. Round-trip is measured and displayed as the number of concurrent users increases. These metrics were collected using the Grinder tool on a tuned VM that represents a typical system under load. As can be seen, the proposed new solution tracks closely in performance with existing conventional approaches. A minor constant overhead is identified, and can be explained by the increased level of internal inter-component messaging required to implement the increased level of message flow required by the new system. This constant overhead is comparable to overheads observed in more complex traditional websites, where the login function is more complicated than a simple relational database lookup.

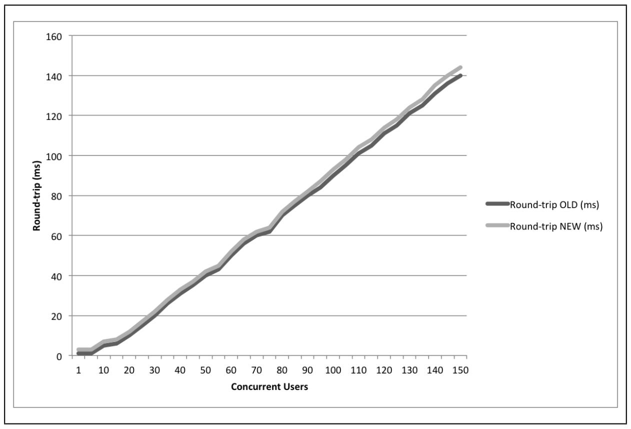

Figure 3. Round-trip comparison

Figure 4 presents comparison between the respective total bytes transferred for each architecture. Each value plotted is the sum total of all bytes transferred between all components in the experiment. Transfer of bytes is measured between the following points in the communication flow:

- Web browser and web server.
- Components and the bus.
- The bus and other components.
- Components and the data storage solution.

The comparative total bytes transferred tracks as expected when additional inter-component messaging requirements of the new solution is considered. This increase in volume of traffic is easily absorbed by the proposed new architecture because components can be distributed across multiple environments, and the additional bandwidth requirements accordingly amortised.

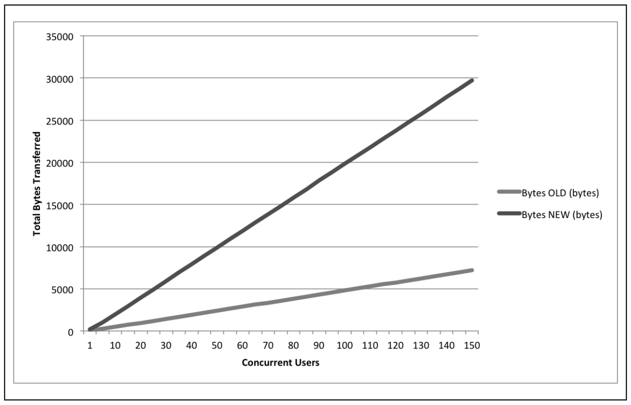

Figure 4. Total-bytes comparison

VII. Conclusion and Future Work

The system architecture presented in this paper continues to evolve as we seek to find the right mix between web application, thick application, and app-store style environments. The ”Super Browser” concept is realised as an abstraction, with the web browser becoming but one component in the overall system. To the software developer, applications are generated as a collection of inter-related components, without the need for the developer to be specifically concerned with how and where the components are executed. The full benefits of a component-based software engineering approach are realised, including the ability for systems to be designed to include off-the-shelf components. The solution presented is backwards compatible with existing web standards because the Web Browser is used as the realisation of the user interface. A Web Browser supporting the new system can seamlessly access both traditional Web 2.0 applications and applications developed using this component-based approach. The implementation is currently being finalised, with the presented metrics suggesting that performance of the new system architecture is no worse than for existing Web 2.0 technologies. The feature comparison summarised in the evaluation section shows that from a feature perspective the new model provides many benefits over existing technologies. The role of the web browser itself is simplified by distributing the hotspots such as code execution and security to a component-based design. The web browser returns to its original role of providing a user interface. This solution provides the additional benefit of allowing applications to transparently be built out of distributed components. These components can execute anywhere the user can access resources - such as on mobile devices, within server farms, or in Cloud Computing implementations.

The following items of future work have been identified:

- Further runtime environment development to cater for programming languages other than Java.
- Enhanced component locality functionality to reduce the reliance on a single component location directory.
- A comparison of security between the new user interface component design and the concept of web browser plugin process sandboxing.
- Enhancements that allow multiple users to share the same instance of a component communication bus.
- Further experimental results, specifically in comparison to existing technologies.

While the implementation of the design during this initial work is in that of middleware, the final implementation will most likely be a shared implementation across middleware and operating system. Current operating systems do not provide a component-based execution model suitable for executing the above described components, but it is envisaged that execution of these components will become a native feature of the operating system in the future.

REFERENCES

[1] WebDevout. (2009, November) Web browser security statistics, http://www.webdevout.net/browser-security. [Online]. Available: http://www.webdevout.net/browser-security. (cited April 2012)

[2] D. Coursey, "Html5 could be the os killer," *PCWorld Business Centre*, 2009.

[3] C. Reis, A. Barth, and C. Pizano, "Browser security: lessons from google chrome," *Commun. ACM*, vol. 52, no. 8, pp. 45–49, 2009.

[4] G. Boss, P. Malladi, D. Quan, L. Legregni, and H. Hall. (2007, October) Cloud computing. [Online]. Available: http://download.boulder.ibm.com/ibmdl/pub/software/dw/wes/hipods/Cloud_computing_wp_final_8Oct.pdf. (cited June 2010)

[5] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599 – 616, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/B6V06-4V47C7R-1/2/d339f420c2691994442c9198e00ac87e

[6] M. Brandel, "The trouble with cloud: Vendor lock-in," *CIO.com*, 2009.

[7] Adobe Systems Incorporated. (2012) Adobe flash. [Online]. Available: http://www.adobe.com/products/flashplayer.html

[8] Microsoft. (2012) Microsoft silverlight. [Online]. Available: http://www.microsoft.com/silverlight/

[9] W3C. (2004) Web services architecture. [Online]. Available: http://www.w3.org/TR/ws-arch/

[10] M. P. Papazoglou and B. Kratz, "Web services technology in support of business transactions," *Service Oriented Computing and Applications*, vol. 1, no. 1, pp. 51–63, April 2007.

[11] I. Hickson and I. Google, *HTML 5 Specification*, editors draft ed., W3C, 2012. [Online]. Available: http://dev.w3.org/html5/spec/Overview.html

[12] D. Flanagan, *Javascript: the definitive guide*. O'Reilly, 2002.

[13] D. A. Chappell, *Enterprise Service Bus*. O'Reilly, 2004.

[14] P.-P. Koch. (2012) Quirksmode w3c dom compatability tables. [Online]. Available: http://www.quirksmode.org/compatibility.html

[15] M. Wallis, F. Henskens, and M. Hannaford, "Publish/subscribe model for personal data on the internet," in *6th International Conference on Web Information Systems and Technologies (WEBIST-2010)*. INSTICC, April 2010.

[16] ——, "Web 2.0 data: Decoupling ownership from provision," *International Journal on Advances in Internet Technology, issn 1942-2652, vol. 4, no. 1 and 2, year 2011*, pp. 47 – 59, 2011.

[17] Mozilla Foundation. (2008) Mozilla - about. [Online]. Available: http://www.mozilla.org/about/

[18] Java. (2012) Glassfish - open source application server. [Online]. Available: http://glassfish.java.net

[19] D. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM, Issue 12*, vol. 15, 1972.

[20] M. Wallis, F. Henskens, and M. Hannaford, "A distributed content storage model for web applications," in *INTERNET 2010*, 2010, pp. 98 – 103.

[21] Oracle. (2011) What is java web start and how is it launched? [Online]. Available: http://www.java.com/en/download/faq/java_webstart.xml