# Web 2.0 Data: Decoupling Ownership from Provision

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*—Current Internet trends have caused us to outgrow existing online data storage paradigms. This paper presents an extended model for distributed online data storage. The model addresses issues of data duplication, data freshness and data ownership, while facilitating two modes of data access - direct and indirect. Direct data access is implemented using advanced handoff techniques while indirect access is implemented using robust server-to-server protocols that enforce strict policies on data management. Results are presented that compare this solution to existing technologies and an example migration path is described for existing Web 2.0 applications wishing to adopt this new paradigm.**

***Keywords*-distributed, storage, personal data, data ownership**

## I. Introduction

The current data storage model for Web 2.0 applications defines data stores managed by the web application owner. Web 2.0 has increased the popularity of user-generated content, which has placed massive quantities of data into these stores. Users are often forced into signing EULAs that restrict their ownership of this data. In fact, having the application provider manage this data has resulted in problems with data ownership, data freshness and data duplication. Previous work [1], [2] introduced the concept of a model that vests content storage with the original content generator. This allows the content generator to retain ownership while supporting most Web 2.0 applications ability to function as they do currently. The model creates the possibility of a single-version-of-the-truth for user-generated content.

The Distributed Data Storage Application Programming Interface (DDS API) allows web applications to seamlessly present user-generated content to a 3rd party user. Using this API, 3rd-party users interact directly with both the web application and the DDS systems hosting the end user's content. Data owners can manage their data elements by interacting with a DDS directly while also exposing this data to web applications via a publish/subscribe model. This paper presents version 2 of the Distributed Data Service (DDS) API. In version 2, information can be accessed via a pass-through mode, which allows 3rd-party web browsers direct access to remote DDSs. Additionally, web applications may request information directly from DDS services under strict contractual arrangements before enriching the data and providing it to the end user. This direct access is protected by strict electronic contracts and dynamic handshaking.

The justification for a distributed storage model is based on the concept that in a Web 2.0 application the majority of information is generated in a distributed fashion by end users. The term 'Web 2.0' is a business generated term, which can be traced back to 2005 when O'Reilly first defined the concept of web generations [3]. At the time, Web 2.0 was identified as any web application that matched the following criteria:

- The application represents a service offering and is not pre-packaged software.
- The application data evolves as the service is used. This is in contrast to applications in which static data is generated solely by the application owner.
- A framework is provided that supports and encourages user submission of software enhancements. These submissions are generally in the form of plug-ins or extensions to the web application.
- Evolution of the application is driven by the end user as well as the application owner.
- The application interface supports interaction from multiple client devices such as mobile phones and PDAs.
- The application provides a lightweight, yet dynamic, user interface.

A high-level overview of the Web 2.0 design is shown in Figure 2.

A key criterion of interest to this research is that evolution of the website is tied to the degree of user interaction. This is driven directly by the fact that the primary service provided by a Web 2.0 site generally relies heavily on user-generated content. The more that data owners interact with the website, the greater the experience of all users. This paradigm has proven popular because website owners are no longer solely responsible for content generation. The fact that the amount and richness of the data provided by data owners can be directly tied to the success of a website only exacerbates the problem of data ownership, as the contained data becomes an important asset for the website owners. Without this asset, they would have substantially less to offer to their user base.

This paper formally defines the interface used by the distributed data service - the DDS API version 2. Section II provides some background information on where this

research sits in our larger work. Section III follows with an overview of the problems which the DDS addresses. Section IV provides an overview of existing approaches to this problem. Section V formally defines the DDS APIv2 as a specification that can be used to implement distributed data systems using a distributed content management model. It encompasses all phases of the data management including the insertion and retrieval of data by 3rd parties. Section VI reviews the strict contractual model used in enriched-mode. Section VII describes the proof-of-concept implementation provided by this paper, which includes multi-language and multi-platform components. Sections VIII and VIII-C present performance metrics and provide a comparison of the DDS solution against other existing technologies. Section IX provides an overview of how the solution scales before Section X presents an overview and conclusion.

A preliminary version of this work has been reported [1].

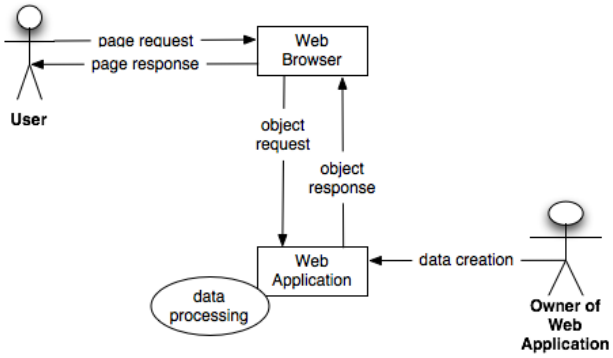


Figure 1. Web 1.0 model

## II. Background

Web 1.0, as shown in Figure 1 web sites comprise service providers creating and presenting static content for consumption through users' web browsers. The move to Web 2.0 has seen an increase in focus on user-generated content and dynamic user interfaces. Support for these concepts was never built into the original web browser design, which was focused primarily on static content [3]. Security was also added as an afterthought, quite unsuccessfully, as can be seen from the large number of security alerts related to web browser technology [4]. While attempts have been made to improve the browsers, often using complete rewrites [5], it has become apparent that a re-design of the way we interact online is required.

The advent of Cloud Computing [6] has brought about change to the server-side of the Web. The deployment of Web 2.0 applications in the Cloud has raised concerns to do with data security and regional regulations [7]. Vendor lock-in has also become a major issue [8] with vendors refusing to share common APIs.

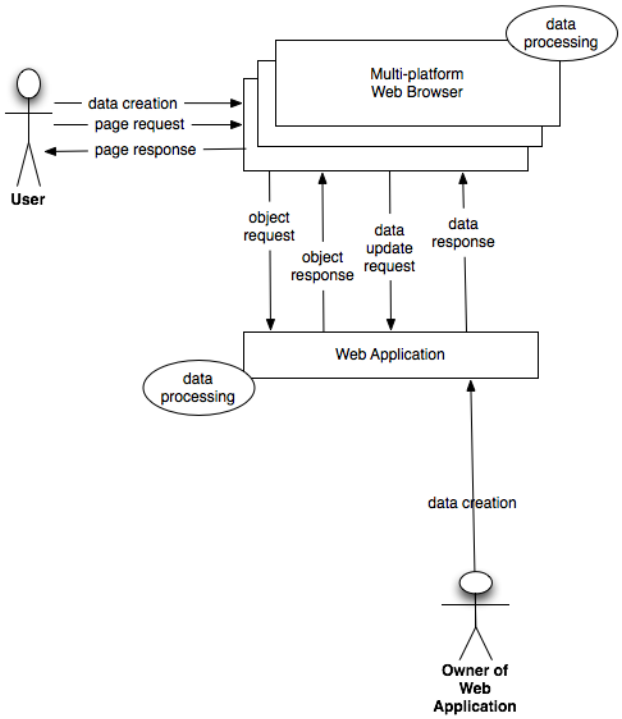


Figure 2. Web 2.0 model

The authors believe that the roles of Internet-connected computers and software should be re-examined. This re-examination suggests that the concept of the Cloud should be expanded to encompass both server and client side resources [9]. Developers can then use component-based software engineering tools to develop applications built from various components that execute in a distributed fashion [10].

The distributed component model raises the question of where data should be stored on the Web. The DDS API allows data storage to be distributed across the Internet while decoupling ownership and management of data from the provision of web applications. The data owner retains access and control over security of their data while continuing to allow the current generation of web applications to function with minimal change.

## III. Problem Description

The problems with the existing storage models in Web 2.0 can be viewed from two aspects - the web application and the end user. From a web application viewpoint, monolithic storage structures have resulted in the following key problems:

1) Web applications must host and manage large-scale database systems to store all the user-generated content in their data centre.
2) Since all the data is stored by the web application the hosting network must bear the full load of transferring

that data between the monolithic storage structure and the user-base.

3) Web application providers must deal with complex privacy and regulation issues stemming from the fact that access to user-generated content can sometimes be restricted by privacy laws.
4) Monolithic storage models introduce large single-points-of-failure. In a Web 2.0 model, the risk is that a single Web application hosting valuable data may go offline, either temporarily or permanently.

Three key problems affecting the end user are data ownership, data freshness and data duplication [2].

1) Data ownership is the most important issue of the three. This relates to the fact that 3rd party web applications can currently place restrictions on the usage of a user's data just because they store it locally on their servers. Data owners are forced to agree to EULAs to use services that can take away the owner's rights to their own data, despite them owning the original data [11].
2) Data freshness refers to the situation in which a user provides data to one or more web applications and that data changes at a later date. The original data provided to the web applications then becomes stale, unless the user is able to recall and update all web applications to whom the data has been provided. The manual user update would be performed on a per-web-application basis, possibly dealing with access issues such as expired login or forgotten credentials.
3) Data duplication refers to the situation in which multiple web applications store copies of the same piece of data. While the associated implementations seems trivial when considering such pieces of data as a single postal address, the issue expands dramatically when dealing with multimedia data such as image, music and video libraries.

These issues have all developed through the increased use of SAAS and Web 2.0 architectures. SAAS (Software as a Service) is a model where software is provided as an online service as opposed to a distributed executable piece of code. While the benefits of a SAAS model are well documented, it is the combined use of SAAS with the increased level of user-content being stored online, which has led to the above problems. The model presented in this paper resolves these issues while also maintaining the benefits of SAAS and the Web 2.0 model.

## IV. Literature Review

Web applications treat user-data as as a commodity [12] and lack the motivation to relinquish control. However, it is in the user's best interest to control their data and privately manage the three key issues of data duplication, data freshness and data ownership.

Much can be gained by reviewing distributed storage in general. Technologies such as CORBA [13] have long leveraged distributed storage concepts. Such ideas apply across a wide range of technologies from Grid Computing [14] to Distributed Operating Systems [15]. The Internet provides a global transport mechanism and is therefore a perfect environment for deployment of distributed systems. Ongoing advances in communications technology can only assist.

Distributed Storage in Cloud Computing [16] provides storage transparently across multiple devices and sites. End users, even application developers, are unaware of where data is stored. The Cloud approach provides developers with access to large managed data storage operations requiring minimal effort. Ongoing concerns include trust, security and regulatory obligations [7].

Research relating to storage on the client side has focused on Local Storage [17], as introduced in HTML5. Local storage allows web applications to utilise basic key/value style storage, which is implemented within the users web browser. This represents an evolution from Cookies [18]; the focus on provision of enriched user interfaces with decreased bandwidth requirements. HTML5 continues to be an emerging technology, and criticism of its viability to solve the current issues with Web 2.0 are common [19].

Software engineering, operating system and Cloud Computing research combine to define the concept of a Web Operating System. This concept was originally introduced as a way of deploying an operating system that was capable of managing distributed resources [20]. As it evolved the focus moved to provision of remote online desktops [21] and bridging the gap between the Operating System and the Semantic Web [22]. The primary driving force in the marketplace is currently Google with their Chrome OS, a replacement for traditional operating systems. Chrome OS promotes access to web applications using the web browser itself as a large component of the operating system [23]. Increasing the scope of the web browser's involvement in application execution has been introduced in many forums. The Super Browser [24] focuses in this area and expands the design of the classic web browser beyond the requirements of Web 2.0.

Analysis of the above technologies has identified the need for a consolidated approach to storage across web applications. This becomes especially important when we expand the scope of web applications to support components executing on both the client side and the server side. This paper introduces a model that addresses this need.

## V. Model: DDSv2 API

The model presented in Figures 3 through 7 addresses the identified problems of data ownership, freshness and duplication. This is accomplished by introducing additional technology that allows storage of data to be offloaded

from the web application. Instead, the storage is made the responsibility of 3rd party storage providers. These providers lease storage services to individual data owners, and act as a 'single version of the truth' provider for that piece of data. Data-owners can either be corporations, business groups, public entities or even individuals. Enhancements to the standard web browser design allows this data to be accessed seamlessly for integration into displayed web pages. An API is established to govern communication between the various actors in the model. These web browser enhancements are a stepping stone between existing browser technologies and a complete Super-Browser implementation [25].

The model definition can be broken into three main areas: storage, access and presentation.

### A. Storage

The first phase of distributing data storage involves the data owner subscribing to a distributed data service (DDS). This service is responsible for storing the owner's data elements and is located either in-house or outsourced to a specialised data storage provider. The subscription procedure is shown in Figure 3.

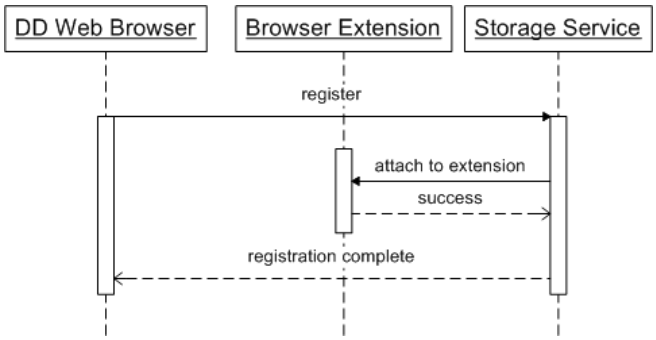


Figure 3. SS Registration

To aid integration of the DDS into the data owner's web experience a new module is introduced called the 'DDS browser extension'. This extension executes within the data owner's web browser, and is aware of the new distributed data service model. When a Web 2.0 site requests content from the data owner, he or she provides a link into the DDS in which the data is stored. Later, when the data is needed for display as part of a page generated by the site, the displaying browser instance uses the embedded link to directly retrieve the data from the owners's DDS. In current pilot implementations of the model, the module is implemented using the browser-extension technologies provided by most modern web browsers [26]. Future Super-Browser implementations will see the module as a distinct component executing within the web browser virtual machine [25]. Communication between the browser extension and the storage service is achieved by inserting a *DDS-StorageService-AttachRequest* into the HTTP response. This request is transparently inspected on-the-fly by the browser extension, which allows the extension to re-write parts of the response HTML dynamically. For the data-owner, the extension re-writes HTML form fields with links that activate a browsing interface for selecting data objects. For the end user, the extension re-writes links into a remote DDS with data returned from that specific DDS.

Implementation of the storage service registration process and the browser extension is implementation specific, though the API for linking the two components together is the first aspect defined by the DDS API. As long as implementations maintain the API for the DDS series of request/response messages, interoperability is maintained. The security of the connection between the browser extension and storage service is also implementation specific, but it is assumed that SSL encryption would be used at the tunnel level and basic authentication would be used to authenticate the end user to the storage service.

Phase two of the process, presented in Figure 4, involves the data owner publishing content to the storage service. Again, the implementation is not restricted by the API as long as each piece of stored data is given a unique identifier that is global in that data owner's domain. The unique identifier comprises three components:

*[name]:[path]@[system]*

The *name* component represents an identifier for each specific piece of data (for example, credit_card). The *path* component supports a hierarchical storage structure allowing a data owner wishing to store various groupings of data (for example, a data owner may have separate sets of 'personal' and 'business' data). The *system* component is a unique identifier for the specific distributed data service. It is generally a DNS name referencing the DDS itself. To reduce vendor lock-in it is recommended that data owners implement their own DNS pointers so that migration from one DDS to another does not result in the reissue of unique identifiers due to a change in the related system component. Using DNS for the system component also solves the issues of locating a specific entities DDS.

The data format proposed by this design is based on a published set of XML schema's that represent each type of data stored by the DDS. While enforcing data format appears restrictive, it is necessary to ensure that interoperability is promoted between web applications and storage services. Such generic interoperability is one of the main requirements for a solution that does not promote vendor lock-in.

### B. Access

Once the data owner has uploaded data into their DDS, the next stage is to allow web applications to subscribe to this data. This is presented in Figure 5. During the registration process for a DDS-enabled web application, the web browser extension inspects the HTTP response traffic from

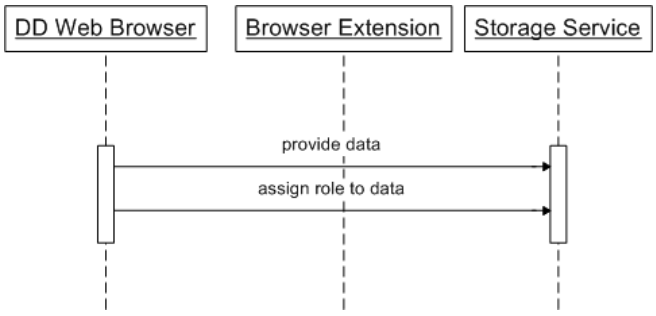


Figure 4. SS Information Upload

the application and detects a *DDS-Application-Subscribe-AuthRequest* request. This request, and the corresponding response generated by the browser extension, is the basis for establishing a trust relationship between the web application and the distributed data service. The request/response messages are built on extensions to SAML [27] and act as a basis for exchanging public-key cryptography credentials between the end user and the web application.

Once a relationship is established, as shown in Figure 6, the data owner establishes a link between their data element and the web application. This link is akin to the data owner uploading content to the web application in a standard Web 2.0 scenario, except that in the DDS design the data owner provides the unique identifier of the data as opposed to uploading the content itself.

The browser extension again plays a key role, ensuring that web applications can support both DDS-enabled and classic Web 2.0 clients. Additional DDS-enabled attributes are inserted into HTML *input* tags to inform the browser extension that the data owner's interface should be modified to accept a unique ID value rather than actual data. This linkage maybe optionally implemented by showing a pop-up window containing an index of the data stored in the DDS to allow the data owner to select individual pieces of data graphically, rather than manually entering the unique ID of the data.

Once the link is established between a piece of data referenced by the web application and the storage location for that data in a DDS, the web application is free to request the data directly from the DDS. This is useful in the circumstance where the web application is still required to store a subset of data locally in order to provide a service. For example, in the case of an image, the web application may request and store meta-data relating to the size of the image to assist in rendering pages during the presentation stage. This also opens up the possibility of web applications temporarily caching data, if permitted by the data owner policy as described in section VI.

### C. Presentation

The final stage of the design is the presentation stage. This stage can execute in two modes. The original DDS v1 provided a single pass-through mode of operation, where the web application hands off responsibility for requesting user content to the web browser. The web browser receives a handoff request and communicates with the various DDSs directly.

An additional mode of operation has been introduced in DDS v2; it allows web applications to operate as clients of a DDS. This 'enriched' mode is useful when the web application is providing a value-added service that cannot be provisioned by the client.

Examples of where each of these modes of operation would be implemented are given in section VII.

*1) Pass-through Presentation:* The presentation stage in pass-through mode is depicted in Figure 7. Here we define how the data is transparently presented to end users. Again the browser extension plays a key role. In this instance the extension operates as a 3rd party and does not have any direct relationship to the DDS of the rendered data. For example, an end user may access a web application and request to view an image collage built from images stored in multiple DDSs owned by multiple data owners.

In this case the web application, instead of returning raw data, will return a *DDS-Present-DataRequest*. This call contains security information exchanged between the web application and DDS during the initial authentication request. This security information is protected using public key cryptography to ensure that it cannot be abused to falsify links between a DDS and unauthorised web applications and clients. The trust relationship enforced in this case is between the web application and the DDS, hence the DDS itself does not need to be aware of all the end users who can render the data linked to a specific web application. The *DDS-Present-DataRequest* message triggers a handoff of the user from the web application to the DDS, allowing the browser extension to request and render the data directly from the DDS, under the web application's instruction.

Under the DDS model, clients are required to perform additional processing to pass-through and cater for the DataRequest messages. Actual data transfer and rendering functions remain largely unaffected other than the fact that the web browser, on average, would be compiling single pages from multiple data sources. These data sources would be a combination of static data from the web application and dynamic data sourced from one or more DDS systems. Web page rendering engines in modern web browsers already support rendering a single page from multiple components so actual page rendering will appear identical to the end user when compared with current solutions. The rendering engine needs to be mindful of handling 'partial' data outages where a subset of DDS's are unavailable.

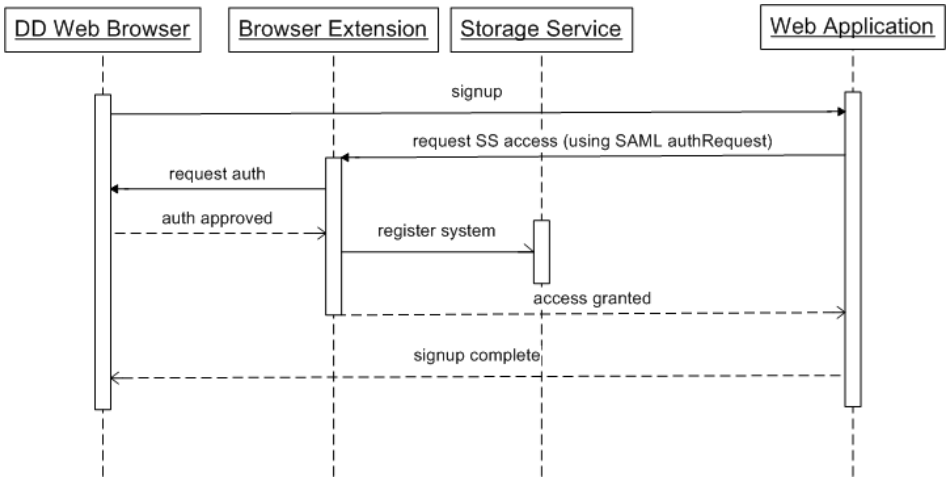


Figure 5. Web Application Registration

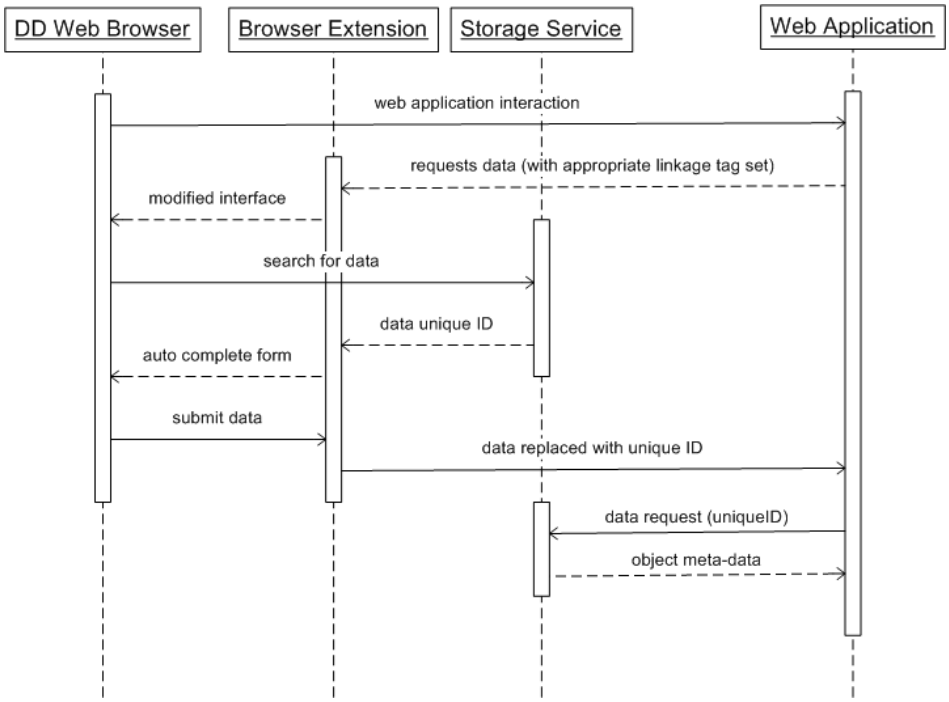


Figure 6. Data Linkage

*2) Enriched Presentation:* The presentation stage in enriched mode is depicted in Figure 8. A web application requests information directly from a DDS, with the intent to enrich the content before presenting it to the end user.

The same *DDS-Present-DataRequest* call is used as seen in pass-through mode. The request is passed directly from the web application to the DDS instead of being passed through the end users' web browser.

A benefit of this model is that end user web browsers are not required to perform any additional processing functions. With this in mind, enriched mode can be seen as a migration strategy or fall-back scenario. In cases where a web browser does not support the DDS API the content can be proxied through the web application using enriched mode.

## VI. Policies and Security

One of the key enhancements to v2 of the DDS API is the introduction of policies. Policies are defined by both web applications and data owners. Both policies must be compatible for a data linkage to occur. Policies enforce rules that define what web applications can do with user data once they have successfully requested a link to that

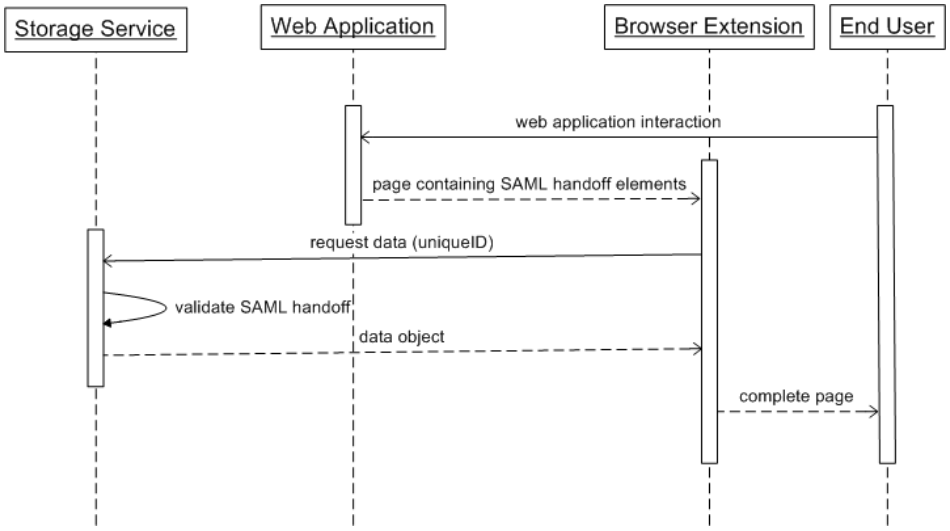


Figure 7. Data Presentation - Pass through

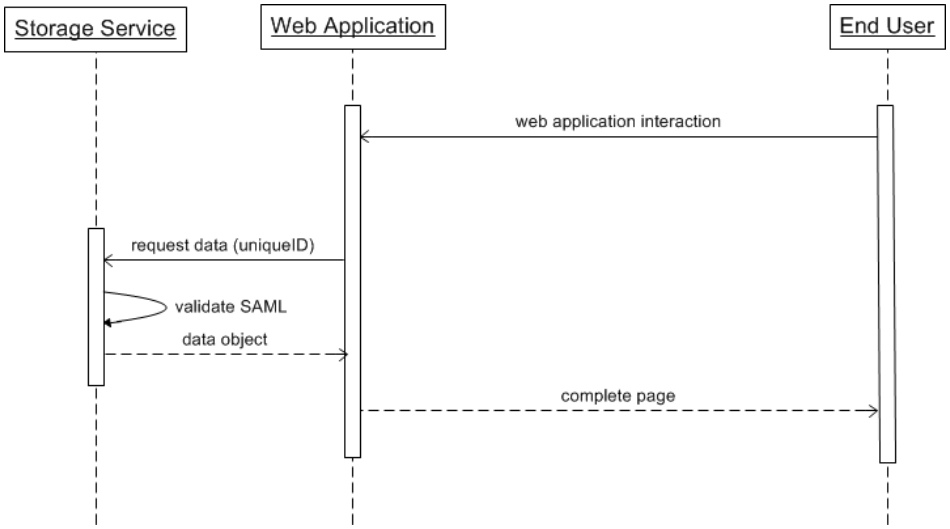


Figure 8. Data Presentation - Enriched

data. In pass-through mode the policy defines the operation required to allow end users' web browsers to obtain data after a successful handoff. The introduction of enriched mode requires web applications to state their intentions before directly accessing DDS data.

Web Application policies are presented to the data owner during the data linkage phase. They can range from a free-text description stating the applications intent to a strict XML schema stating common use-cases such as;

- Transparent pass-through only.
- Retrieve, manipulate, transfer.
- Retrieve and cache for a pre-determined period.
- Retrieve, store and transfer ownership.
- Forward to 3rd party.

If the free-text form of policy is attached to the data linkage request then the data-owner will be presented with the free-text to review before accepting the linkage. If the XML schema approach is used then the policy comparison can be processed automatically using a policy defined by the data-owner. During DDS registration the data owner can create their own policy stating which use-cases they would accept for the data they store in the DDS. This can then be compared automatically to the policy presented by the web application. If the policies fail to match then the data-owner will be given a choice whether to override the policy matching or deny the data linkage request.

Accepted policies form a contract between the data owner and web application. This can be viewed as a capability [28] in a distributed system model. If a web applications policy changes in the future all capabilities must be revoked or

multiple policy versions must be maintained.

These policies form a key foundation for limiting the scope of data freshness, data duplication and data ownership problems. They provide the means for Web Applications to support enhanced user control of the data, which is incorporated into their application. This is in contrast to the existing Web 2.0 model where the data owner is forced to agree to terms dictated by the web application owner, relinquishing all rights to their data. Using the XML schema approach a data owner can publish what policy requirements they enforce on their data in a parseable fashion. This allows for policies to be negotiated and automatically enforced without user interaction. A well defined schema of common use-cases has been defined as part of this API.

In cases where multiple web applications are providing the same service the end user can compare policies and decide which provider offers the most compatible set of requirements. In situations where a web application provider is operating as a monopoly the benefit of policies is centered on providing end users a clear definition how their data will be treated without having to manually read a complex EULA. The authors recognise that a web application may require a user to relinquish rights to their data, however the DDS v2 provides alternatives that previously did not exist.

If the data owner at any stage wishes to revoke access to their data then they can access the DDS and revoke that web applications linkage. This can be done for specific pieces of data or all data stored in the DDS. Is required, the DDS will then perform a server-to-server call to the web application informing it that the data linkage has been invalidated. This will ensure that the web application will not try and handoff a request to that data in future sessions.

## VII. Prototype and Use-Case Experiments

A proof-of-concept prototype was developed to demonstrate all of the components described in section V. The prototype comprises the following:

- A skeleton distributed data service implemented in Java and utilising the Amazon S3 service for data storage.
- A proof-of-concept DDS-enabled web application, implemented in PHP, which is capable of subscribing to a DDS and linking image and postal address data.
- A Mozilla Firefox web browser extension implemented in Javascript to provide the data owner and end user experience.

Multiple programming languages were selected for the prototype to demonstrate the programming language-agnostic nature of the DDS API.

The prototype has been used to demonstrate three use cases. These use-cases are presented below.

### A. Use Case 1 - basic pass through

The domain of this use-case is an in-house group address-book application with which users can create a profile and upload their office address and a profile photo. The address and photo data are stored in the data owners DDS. The following is an example runtime flow from the prototype application. It describes a user linking some data and a second user in turn rendering that data.

- User A (Bob) accesses the website for his DDS of choice and begins the registration process.
- DDS(Bob) sends an attachment request message (through the HTML response) that is detected by his web browser extension.
  - *DDS:DDS-StorageService-AttachRequest() to Extension(Bob)*
- Extension(Bob) requests Bob's approval to attach to the DDS and sends a successful response message.
  - *Extension(Bob):DDS-StorageService-AttachResponse(SUCCESS) to DDS(Bob)*
- Bob then continues to interact with the website presented by the DDS to upload his office address and profile image data.
- Bob now accesses the website for the address book application (WebApp) and begins the registration process.
- The Web Application sends an attach request to the browser extension in Bob's browser. The request is signed with the web application's private key and includes a copy of the web application's public key for verification.
  - *WebApp:DDS-Application-Subscribe-AuthRequest(publickey(WebApp),WebApp) to Extension(Bob)*
- The browser extension requests Bob's authorisation to allow that web application to subscribe to data within his DDS and sends back a response. The browser also forwards the request on the DDS so the DDS can locally register the request.
  - *Extension(Bob):DDS-Application-Subscribe-AuthResponse(SUCCESS) to WebApp*
- The web application then allows Bob to upload an image. Attached to the standard INPUT HTML element an additional DDS-Enabled="true" attribute is included. This instructs Extension(Bob) to render that input element as a DDS data-lookup field.
- Bob selects his profile image from the pop-up DDS interface and the browser extension provides the unique ID of the data ID(image) back to the web application for storage.
- User B (Alice) now accesses the website for the address book application and asks to view Bob's profile.
- The web application inserts a data request message into the HTML response that is received by Extension(Alice).
  - *WebApp:DDS-Present-DataRequest(publickey(WebApp),ID(image)) to Extension(Alice)*

- Alice's web browser extension then establishes a direct connection to Bob's DDS using the system code provided in ID(image) and forwards on the data request.
- DDS(Bob) authenticates the request by validating the signature of the message using the publickey(WebApp) established during the authentication request stage.
- DDS(Bob) then returns the image for rendering by Extension(Alice).

### B. Use Case 2 - enhanced pass through

This use-case presents the model where data enrichment is required but the client-side is capable of providing the processing. A classic example of this is providing a map of a users address.

The DDS registration and web application phases are the same as presented in the first use case. The data linkage and presentation layers change as follows;

- The web application allows Bob to upload his address. Attached to the standard INPUT HTML element an additional DDS-Enabled="true" attribute is included. This instructs Extension(Bob) to render that input element as a DDS data-lookup field.
- Bob selects his address from the pop-up DDS interface and the browser extension provides the unique ID of the data ID(address) back to the web application for storage.
- User B (Alice) now accesses the website for the address book application and asks to view Bob's profile.
- The web application inserts a data request message into the HTML response that is received by Extension(Alice).
  - *WebApp:DDS-Present-DataRequest(publickey(WebApp), ID(address)) to Extension(Alice)*
- Alice's web browser extension then establishes a direct connection to Bob's DDS using the system code provided in ID(address) and forwards on the data request.
- DDS(Bob) authenticates the request by validating the signature of the message using the publickey(WebApp) established during the authentication request stage.
- DDS(Bob) then returns the address for rendering by Extension(Alice).
- The address is passed by Extension(Alice) to the web browser where it is received by client-side code such as a Javascript library
- The Javascript library takes the address information and passes it to a 3rd party web service, which returns a graphical representation of the address as a map.

In the above use-case it is expected that DDS(Bob) would have enforced policy on the request when it saw that the WebApp was requesting the linkage. The DDS would be aware that the data would be eventually on-forwarded to a 3rd party (the map generating service) by inspecting the requested policy. This would allow Bob to limit the scope-of-use of his address information.

### C. Use Case 3 - server-side enrichment

This use-case presents an example where pass-through is not sufficient and the web application requires direct access to the end users' data. In this scenario the user stores their address information and the web application needs to directly access the DDS so that it can obtain content for a mailing label.

Again, the DDS registration and web application registration phases are the same as presented in the second use case. The data linkage phase and presentation phases change as follows;

- User A (Bob) returns to the website where he has previous registered and purchases an item.
- The web application requests that Bob provides his address. Attached to the standard INPUT HTML element an additional DDS-Enabled="true" attribute is included. This instructs Extension(Bob) to render that input element as a DDS data-lookup field.
- Bob selects his address from the pop-up DDS interface and the browser extension provides the unique ID of the data ID(address) back to the web application for storage.
- The web application processes the request and readies the item for shipment. The application then makes a request directly to DDS(Bob) requesting the address.
  - *WebApp:DDS-Present-DataRequest(publickey(WebApp), ID(address)) to DDS(Bob)*
- DDS(Bob) authenticates the request by validating the signature of the message using the publickey(WebApp) established during the authentication request stage.
- DDS(Bob) then returns the address for use by the web application.

In this case, there is no 3rd party and the web application itself requires access to the information stored in the DDS. The web application does not permanently store the address information as it may become stale, hence ID(address) is stored instead, and is used to re-request for future orders.

## VIII. System Evaluation

### A. Functional Requirements

The model presented in this paper sets out to solve multiple issues stemming from the traditional monolithic storage approach used by web applications on the Internet.

Distributing data element storage greatly reduces the resource requirements of web applications. Storage requirements will decrease to only those needed to store meta data on the web application itself rather than the user-generated content. Bandwidth requirements for the web application will drop as the application will only be returning basic

HTML, CSS, script and meta-content such as logos and branding. All user-generated content will be directly transferred to the end user from the related DDS systems. Lastly, the web application provider will no longer be required to meet varying privacy legislation requirements as they will not be directly storing any user's personal data. This requirement is instead offloaded to the DDS providers, which can operate in the same geographical region (and hence be subject to the same legislature) as the data owner.

From the end user perspective, the model addresses the data freshness issue by replacing the N multiple copies of a data element with N links that point to a single instance of the data stored in the data owner's DDS. These N links are abstracted using DNS technology to ensure that a user can migrate from one DDS to another without invalidating the links. This removes the danger of a system accessing obsolete versions of data by creating a single version of the truth for every data element in the system.

The model also addresses the data duplication-created storage wastage issue. This is true provided the number of bytes used to store a link is less than the number of bytes used to store the actual data. With this assumption we achieve a reduction of the storage requirements in a single system from (M * objects) to (N * objects) where N is the size of a link and M is the average size of stored objects. A web application would only be required to store the [system] component of the link once per user.

Most importantly, the issue of data ownership is also addressed. Use of owners' data was previously dictated by web applications, and was typically enforced by end user licensing 'agreements'. If a user wished to use a particular web application they had no choice but to accept the EULA. With the described DDS model, the user has more freedom. It is safe to assume that the DDS itself may also enforce a EULA on the end user, but in this situation the user has the buying power to procure services from another DDS provider that requests a less restrictive license.

Security of the DDS system is provided in multiple layers. All communication between the data owner and the DDS can be protected using such existing technologies as SSL. Basic authentication would suffice when the transport layer is protected. The link established by the data owner with the web application forms the basis of an authentication token, which is then used to authenticate end users through the web application into the data owners DDS. This handoff is protected using the SAML handoff framework. All security assertions as signed with the DDS validating the signature as belonging to the data owner. This allows the DDS to ensure that any request for data coming from an end user, through a specific web application, has been authorised by the data-owner.

### B. Comparative Evaluation

While the authors could not find any other model specifically targeting the core issues of this paper, there are systems that are similar in nature to the DDS.

*1) CMS:* Parallels can be drawn between the DDS and Content Management Systems [29]. The DDS can be viewed as a personal CMS that allows its content to be seamlessly embedded into 3rd party web applications. The DDS provides distributed storage of user content for web applications that previously relied on monolithic storage repositories.

Current CMS solutions do not scale to the level required to implement an Internet-wide distributed storage solution due to their own reliance on monolithic storage structures.

*2) CDN:* Content Delivery Networks [30] are distributed storage networks that allow companies to host data objects on 3rd party networks. This allows them to take advantage of geo-location based load balancing and link peering to achieve reduced bandwidth costs. The typical CDN solution is similar to the delivery paradigm in the DDS model except that CDNs do not currently provide a seamless way for data owners to push content into the network and have that content transparently accessed by authorised web applications. CDNs are static in nature, and do not scale to the dynamic features that the DDS model provides.

*3) Cloud SSP:* The presented design ties directly into the realms of Cloud Computing [16], Service-Orientated architectures [31] and SAAS (Software-as-a-service) [32]. In a sense, a DDS can be seen as a SSP (Storage Service Provider) in a Storage-as-a-service [33] cloud component that allows other web applications to publish and subscribe to data within the Cloud. The DDS model described in this paper, however, provides the necessary additional access and presentation layers on-top of the storage to ensure that the user experience is seamless.

Cloud computing can play an important part in the design and hosting of the DDS storage system itself. As the DDS API does not explicitly define the internal design of the DDS, the vendor is free to, for example, use Cloud Computing technologies, this providing a DDS solution that benefits from the dynamic scalability and per-usage business models that the Cloud provides.

### C. Performance

The paradigm shift described in this paper dictates a movement of data storage away from classic monolithic storage, towards a distributed network of data storage services. As such, performance has been analysed to identify overheads introduced by the additional access and presentation complexity. While a small constant overhead was identified due to the web application-to-DDS handoff requirements, performance increases were also identified in the following areas:

- Speed improvements under high-load situations due to the reduced data transfer requirements of web applica-

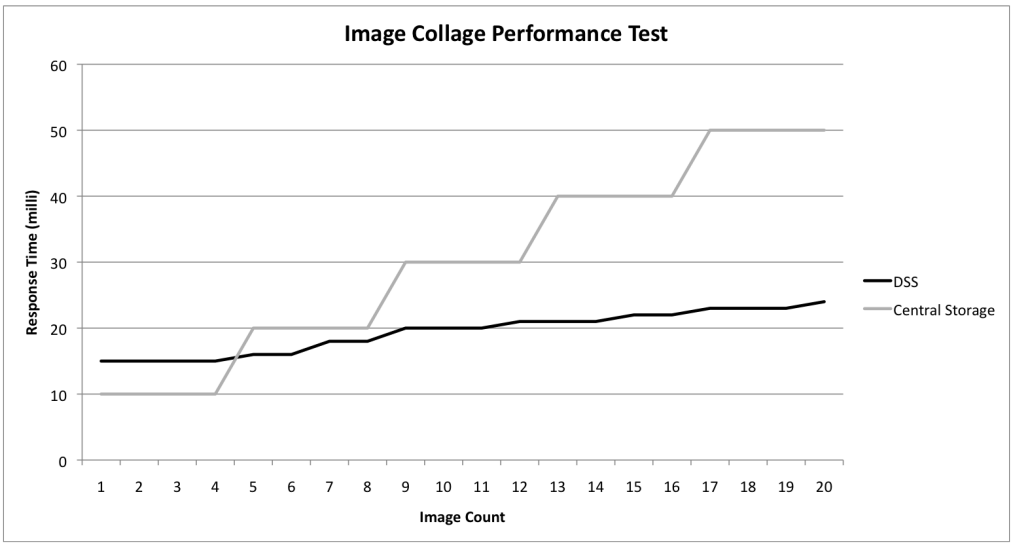


Figure 9. Performance results of the Collage Test showing pipeline enhancements

tions. Instead of the web application being responsible for provision of all the displayed data, the data transfer requirements are shared between the web application and the various linked distributed data services. This has the potential to reduce the network load of web applications.

- Client geo-locality can be utilised when users access data that is geographical in nature and when the required distributed data services are located closer to the client than to the web application. For example, a user browsing images of their friends on a social networking site would experience improved performance if the DDSs for their friends were less network hops away than the social networking web application itself.
- Speed improvements were identified in cases in where a single webpage is built of multiple separately loaded elements, a popular model in systems that rely heavily on user-generated content. In the general Web 2.0 case, browser pipelining restrictions limit the number of simultaneous network requests to any server. By distributing data storage the impact of these restrictions is reduced. Figure 9 shows the performance improvements for the case in which a single page is built from multiple data elements, each separately sourced. The experiment was performed with a pipelining restriction of four simultaneous connections per server. The results show a marked improvement using the DDS system.

Performance of the DDS system can be adversely affected by poor connectivity and bandwidth to specific DDS nodes. The open market for storage services will assist in driving competition between storage providers to reduce this risk. Obviously there are upfront costs involved in the integrating web applications with the DDS API, but these are offset by reduction in ongoing costs bandwidth and storage costs.

### D. Backwards Compatibility and Migration Path

With the introduction of enriched mode a clear migration path has been presented for web applications taking on the new storage paradigm. A browser check can be performed during session initiation that informs the web browser if the end user is using a browser capable of the DDSv2 API. If not, enriched mode can be used to proxy the data to the end user. ”Pass-through” rights can be enforced via policy and users using older browsers will still see the benefits of this approach. Network limitations come into play in this scenario as data is proxied through the remote web application rather than being accessed directly by the end users’ web browser from the various distributed storage services.

## IX. Scalability

The DDS system scales exceptionally well due to the decentralised nature of the data storage. Each user is free to choose their own DDS host(s). As the user base grows, the number of DDSs linked in a web application also grows. Each unique DDS can execute within a Cloud Computing environment, hence internal scalability is also supported in the situation where large numbers of data owners choose to use the same DDS (for example, all users of a particular University may choose to use a University-hosted DDS solution).

The DDS solution also scales into the corporate space where each corporate entity could host their own DDS. This would allow the employees of a company to share data internally, as well as externally through restricted publish/subscribe functions. The openness of the DDS API

allows corporate entities to protect their data by controlling which web applications subscribe to specific pieces of data.

From an end user perspective, the DDS system scales in the same fashion as a traditional client/server model. The transparent nature in the way the DDS browser extension provides visibility of DDS-stored information ensures that the end users' experience remains unaltered. From a connection viewpoint, the constant overhead described in the performance review above has no effect on the solutions ability to scale when compared to traditional approaches.

## X. Conclusion

This paper addresses three concerns resulting from the growing popularity of Web 2.0 applications by formally defining a new paradigm for the distributed storage of data on the Internet. The standard for web applications has evolved, from static pages comprising a limited number of elements to complex pages rendered from a large numbers of elements. Web 2.0 has seen a trend towards bandwidth intensive elements originally generated by end users. As the user take-up of Web 2.0 applications continues, it is sensible to adopt a distributed approach that parallels the way content is originally generated.

Problems caused by the usage of monolithic data storage features have been mitigated by adopting a distributed storage approach. Moving from monolithic to distributed structures is a proven technique for sharing load that has been used extensively in other areas such as Cloud Computing [16] and Transaction Management [24].

The key issue of data ownership is addressed for end users by ensuring that storage is the responsibility of distributed data service(s) directly engaged by them. DDS providers are liable to data owners, not to web applications, and hence data owners have control over use of their data. Data ownership is clear-cut because owners are responsible for both storage of, and access to, the data.

Data freshness is addressed using a publish/subscribe model and an enhanced SAML-based handoff model for data presentation. The data rendered in web pages is always the freshest version because it is sourced directly from the data owner's DDS. Data duplication is also addressed by removing the need for data to be stored by web applications. Appropriate web application registration and linking reduces the number of copies of any piece of data to a single instance stored in the DDS.

Policies implemented during the web application and DDS handshaking ensure that end users are aware of how their data will be treated by that web application. The policies provide a level of protection which does not exist in the current Web 2.0 design. They provide an avenue to comparing multiple web service vendors from a data policy perspective.

Current modelling and experiments show that overall system performance is comparable to the existing Web 2.0 paradigm in the general case, with minor constant overhead caused by the handoff procedures. When a web application renders a page containing multiple data elements from multiple DDS repositories, we observe a performance improvement compared to existing technology due to the bypassing of web browser pipelining restrictions.

Comparisons made against similar systems show that the new paradigm can greatly increase the quality and protection of data in a Web 2.0 space. For the DDS model to become widely utilised the DDS API will need to be adopted as a standard.

## References

[1] M. Wallis, F. A. Henskens, and M. R. Hannaford, "A distributed content storage model for web applications," in *The Second International Conference on Evolving Internet (INTERNET-2010)*, 2010.

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

[3] T. O'Reilly. (2005) What is web 2.0. [Online]. Available: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html. (cited June 2010)

[4] WebDevout, "Web browser security statistics, http://www.webdevout.net/browser-security," *WebDevout*, November 2009. [Online]. Available: http://www.webdevout.net/browser-security

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

[6] 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

[7] H. Newman, "Why cloud storage use could be limited in enterprises," *Enterprise Storage Forum*, 2009.

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

[9] M. Wallis, F. A. Henskens, and M. R. Hannaford, "Expanding the cloud: A component-based architecture to application deployment on the internet," in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2010.

[10] ——, "Component based runtime environment for internet applications," in *IADIS International Conference on Internet Technologies and Society (ITS 2010)*, 2010.

[11] AFP. (2009) About-facebook: backflip on data ownership changes. [Online]. Available: http://www.smh.com.au/articles/2009/02/19/1234632933247.html. (cited June 2010)

[12] T. Lee, "Data ownership might not work for social networking sites," *techdirt*, 2008. [Online]. Available: http://www.techdirt.com/articles/20080516/1124101137.shtml

[13] Object Management Group, *Common Object Request Broker Architecture: Core Specification*, March 2004.

[14] B. Jacob and et al, *Introduction to Grid Computing*. IBM Redbooks, 2005.

[15] A. S. Tanenbaum and R. V. Renesse, "Distributed operating systems," *ACM Comput. Surv.*, vol. 17, no. 4, pp. 419–470, 1985.

[16] 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)

[17] I. Hickson, *HTML 5 Editor's Draft*, w3c editor's draft 2011 ed., W3C, January 2011. [Online]. Available: http://dev.w3.org/html5/spec/Overview.html

[18] D. Raggett, A. L. Hors, and I. Jacobs, *HTML 4.01 Specification*, w3c recommendation december 1999 ed., W3C, December 1999. [Online]. Available: http://www.w3.org/TR/REC-html40/

[19] P. Krill, "W3c: Hold off on deployment html5 in websites," *InfoWorld*, 2010. [Online]. Available: http://www.infoworld.com/d/developer-world/w3c-hold-html5-in-websites-041

[20] A. Vahdat, P. Eastham, C. Yoshikawa, E. Belani, T. Anderson, and D. Culler, "Webos: Operating system services for wide area applications," in *Proceedings of the Seventh Symposium on High Performance Distributed Computing*, 1998. [Online]. Available: http://citeseer.ist.psu.edu/61096.html

[21] R. MacManus, "What is a webos?" *ZDNet Tech Update*, 2006. [Online]. Available: http://blogs.zdnet.com/web2explorer/?p=178

[22] L. Dignan, J. Perlow, and T. Steinert-Threlkeld, "From semantic web (3.0) to the webos (4.0)," *ZDNet Tech Update*, 2007. [Online]. Available: http://blogs.zdnet.com/BTL/?p=4499

[23] S. Pichai and L. Upson, "Introducing the google chrome os," *Google Blog*, 2009.

[24] F. A. Henskens and M. G. Ashton, "Graph-based optimistic transaction management," *Journal of Object Technology*, vol. 6, no. 6, pp. 131–148, July/August 2007.

[25] F. A. Henskens, "Web service transaction management," *International Conference on Software and Data Technologies (ICSOFT)*, July 2007.

[26] K. Feldt, *Programming Firefox: Building Rich Internet Applications with XUL (Paperback)*. O'Reilly Media, Inc, April 2007.

[27] OASIS, "Security assertion markup language (saml) v2.0 technical overview," Working Group, Tech. Rep., 2007.

[28] F. A. Henskens, J. Rosenberg, and J. L. Keedy, "A capability-based distributed shared memory," in *Proceedings of the 14th Australian Computer Science Conference*, 1991.

[29] A. Mauthe and P. Thomas, *Professional Content Management Systems: Handling Digital Media Assets*. Wiley, 2004.

[30] M. Hofmann, *Content Networking: Architecture, Protocols and Practice*. Morgan Kaufmann Publishers, 2005.

[31] M. Bell, *Introduction to Service-Oriented Modeling, Service-Oriented Modeling: Service Analysis, Design, and Architecture*. Wiley and Sons, 2008.

[32] K. Bennett, P. Layzell, D. Budgen, P. Brereton, L. Macaulay, and M. Munro, "Service-based software: the future for flexible software," in *Seventh Asia-Pacific Software Engineering Conference (APSEC'00)*, vol. 17th, 2000, p. 214.

[33] J. Foley, "How to get started with storage-as-a-service," *InformationWeek Business Technology Network*, 2009.