HONOURS PROJECT REPORT

Management System of an AJAX based Digital Repository

Suraj Subrun

sksubrun@gmail.com

Supervisor:

Dr Hussein Suleman
University of Cape Town
Department of Computer Science
hussein@cs.uct.ac.za

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF CAPE TOWN

November 6, 2009.

Mark Breakdown

	Category	Min	Max	Chosen
1	Software Engineering/System Analysis	0	15	5
2	Theoretical Analysis	0	25	0
3	Experiment Design and Execution	0	20	10
4	System Development and Implementation	0	15	15
5	Results, Findings and Conclusion	10	20	10
6	Aim Formulation and Background Work	10		10
7	Quality of Report Writing and Presentation	10		10
8	Adherence to Project Proposal and Quality of Deliverables	10		10
9	Overall General Project Evaluation	0	10	10
Total marks		**80**		**80**

Abstract

Few digital repository systems allow the creation of copies of a collection that can be viewed offline from distributable storage media such DVD-ROMs or flash drives. To date, the system closest to providing such functionality is the Greenstone Digital Library software suite (New Zealand Digital Library Project, 2009). It allows content to be viewed directly off a CD-ROM. However, the problem with Greenstone is that it requires the installation of several software packages including a Web server before this is possible.

This project is aimed at demonstrating the feasibility of a digital repository system supporting the creation of distributable collections that can be used with no prior software installation. This is achieved through the use of Asynchronous JavaScript and XML (AJAX) technologies that are supported by all major Web browsers.

CALJAX is the AJAX-based digital repository system developed. It is broken down into three main components: the browsing and searching components of the distributable collections and the management component of the central repository of the system. This report deals with the central repository management system of CALJAX.

The report discusses the various existing digital repository solutions and their management features. Design solutions of the components of the system implemented are then detailed. The implementation of the management system is described. This was carried out in several iterations and each iteration was subjected to critical evaluations.

The management system implemented is deemed to be an overall success. It supports the critical functionality and qualities expected of a management system. The implementation is scalable, compatible with major browsers and demonstrates good performance and usability.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: H.3.7 Digital Libraries

Keywords

AJAX, Management, Repository.

Acknowledgements

Our supervisor, Dr Hussein Suleman for his help and guidance during the project.

My parents, my sisters and my niece, Layla, for their constant support.

Nawaaz Karimbocus and my other friends for always being there.

All the evaluators of the system for their valuable feedback.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 CALJAX Project Description

1.1.1 Background

There are few digital repository systems that enable users to create copies of a collection that can be distributed on various types of storage media, such as DVD-ROMs or portable flash drives to be viewed offline. To date, the system closest to providing such functionality is the Greenstone Digital Library software suite (New Zealand Digital Library Project, 2009). It allows collections from a digital repository to be written to CD-ROMs which can then be distributed and used for offline viewing. However, the problem with Greenstone is that it requires the installation of several software packages including a Web server before this is possible.

1.1.2 Project Definition and Aim

This project aims to demonstrate the feasibility of a digital repository system supporting the creation of distributable collections that can be used with no prior software installation.

CALJAX is the digital repository system that is developed to demonstrate that such a system is possible. To achieve this, it makes use of Asynchronous JavaScript and XML (AJAX) technologies that are supported by all major Web browsers and hence provide a universal platform with minimal software pre-requisites.

The goal of the CALJAX project is to answer the following research question:

Can AJAX technologies be used to build a digital repository system that supports the creation of distributable collections, extracted from its contents, which have minimal software requirements?

1.2 Project Components

CALJAX was broken down into three main distinguishable parts – the central repository management system, the browsing features of the distributable copies and their searching

capabilities. This report deals with the design, implementation and evaluation of the management system of CALJAX. Figure 1.1 shows an overview of the main components of CALJAX and their interactions.

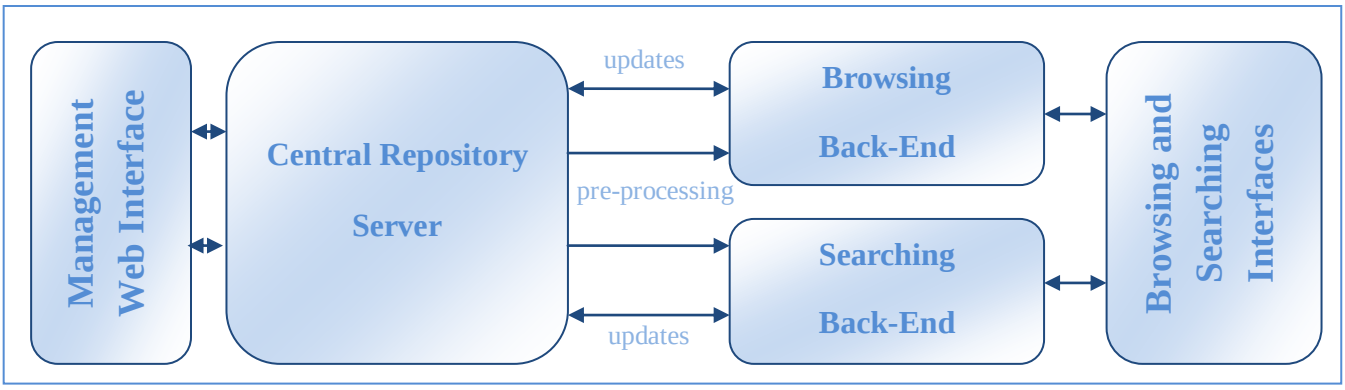

Figure 1.1: Overview of the structure of the whole CALJAX system.

The central repository server stores the items in the repository on its local file system. The browsing and searching back-ends pre-process these items directly for use in their AJAX-based interfaces. The browsing and searching components also communicate with the central repository server to retrieve a list of items that were updated since the distributable collections were created. The central repository server implements the core digital repository functionality and this is used through the management system's Web interface.

1.3 Report Outline

The report begins by looking at the various existing digital repository systems and their design solutions. The critical components of the management system that are implemented in this system are:

- The ability to create, modify and delete repository items, while supporting various types of metadata.
- Providing lists of updated items to distributable copies of the collections.
- An extension system to allow the support of new formats of metadata.
- A Web interface for management, with tested usability.

At various stages of development, the system was evaluated to test its usability, performance and cross-browser compatibility. The system was also tested for scalability on a collection of over 15,000 items.

This report is divided into the following sections:

Chapter 2 Background

This chapter provides an evaluation of existing digital repository systems.

Chapter 3 Design

The scope and limitations of the project are discussed and design solutions for the various components of the system are provided.

Chapter 4 Implementation

The platform and tools used to implement the system are discussed in this chapter. It also provides details of the various stages of the iterative development process. It contains specifications of the communication protocols used in the system.

Chapter 5 Evaluation

This section outlines the results of the various methods used to evaluate the system.

Chapter 6 Conclusions

The final chapter discusses the success of this project and possible improvements to the system as future work.

Chapter 2

Background and Theory

This section provides an evaluation of existing digital repository systems and describes how they work and how they are managed. Investigating the features of existing systems helped produce the list of essential features that would have to be implemented and these are listed. It then considers the recent trends in Web development, often referred to as "Web 2.0", and provides an overview of the AJAX technologies involved in achieving these.

2.1 Current Digital Repository Systems

Ideally, an AJAX-based digital collection would be run straight off the storage medium with a standard browser as the only software requirement. To date, the system closest to providing such functionality is the Greenstone Digital Library software. Greenstone currently allows users to browse digital repository content directly from a CD-ROM but requires a prior installation of several packages before this is possible suite (New Zealand Digital Library Project, 2009). Other important digital repository software tools include the Fedora repository management system, DSpace (The DSpace Foundation, 2009) and EPrints (EPrints, 2009). These systems do not feature the creation of distributable copies of information.

2.1.1 Greenstone

Greenstone is described as providing a way of organising information and publishing it on the Internet in the form of a fully-searchable, metadata-driven digital library (New Zealand Digital Library Project, 2009). Before it can be used, several software packages need to be installed and configured, including a Web server (New Zealand Digital Library Project, 2009). The Web server is needed even when using Greenstone CD-ROMs.

Greenstone works by ingesting metadata (supporting various standards such as Dublin Core, RFC 1807, etc) and various types of digital resources, using different plug-ins for various document formats, to produce its own set of XML data files (New Zealand Digital Library Project, 2009). Plug-ins are written in the Perl scripting language. These are subsequently converted into searchable indices (New Zealand Digital Library Project, 2009). Greenstone provides a Java applet interface called Librarian to handle the management of the repository.

This includes creating collections, defining and editing metadata sets for the collection, reviewing and editing metadata and configuring the system (e.g., configuring the plug-ins).

2.1.2 Fedora

Fedora uses a compound digital object model that aggregates one or more content items into the same digital object (Fedora Development Team, 2005). The Fedora repository system also allows the storage of digital objects relationships which are important for organising objects into collections for management or to simply store semantic relationships between objects (Fedora Development Team, 2005). Fedora has several pre-requisite software package requirements such as a Java runtime environment and a database system. It does not allow the creation of distributable copies of its collections.

The Fedora repository system consists of three layers: the Web Services Exposure Layer, the Core Subsystem Layer and the Storage Layer. The Web Services Exposure Layer provides separate interfaces for management and access of the digital objects in the repository. Fedora has two interfaces for its management. It has the "Fedora Administrator" which is a standard application providing the various functions to ingest, search for and retrieve, modify and purge objects. There is also a "Fedora Web Administrator", which is still in development, that supports these various functions through a Web interface. The Core Subsystem Layer is responsible for the management and access subsystems. The management subsystem implements the operations needed for creating, modifying, deleting, importing, exporting and maintaining digital objects. It also caters for validation and integrity of data. The access subsystem provides the methods for showing the content of digital objects. The Core Subsystem Layer also consists of a security subsystem with policy management and enforcement. The last layer, the storage subsystem, deals with reading, writing and removal of data from the repository. Digital objects are stored as XML-encoded files conforming to the METS schema (Staples, Wayland & Payette, 2003).

2.1.3 EPrints

EPrints also has several software pre-requisites and does not support distributable collections (EPrints, 2009). EPrints provides one of the best Web interfaces amongst the systems that were reviewed, particularly in terms of usability. This interface allows users to list, search through and manage the repository contents. An interesting feature of EPrints is the support for RSS feeds for various purposes, e.g., it is possible to get an RSS feed of the latest journal publications from a particular research group. Search results can be exported to many common formats such as Dublin Core, METS or MODS. Logging into the system allows users to manage all the items they have created. Items can be converted to various other formats. It also allows importing and exporting items of various different formats and from different Web services through the use of plug-ins.

2.1.4 DSpace

DSpace also has several installation pre-requisites and does not feature distributable collections. A study conducted by Körber and Suleman confirms the existence of substantial usability problems in the installation and configuration of DSpace (Körber & Suleman, 2008). An interesting aspect of DSpace is that each item has a bitstream containing data defining the digital object but, in addition, has an associated bitstream format. This is intended to cater for the preservation of data (The DSpace Foundation, 2009). Bitstream format data is more specific than MIME types or file suffixes as, for example, *application/ms-word* or *.doc* files can contain different formats depending on the version of Microsoft Word the file was created with. Furthermore, each bitstream format has a support level to indicate how well the hosting institution is likely to preserve content in the format in the future (The DSpace Foundation, 2009). DSpace also provides access control to some of its features.

DSpace provides a Web interface to allow browsing and searching through the repository contents. This interface also allows for the management of the repository. Metadata types can be specified using schemas. Fields from the schemas can then be used to customise searching and browsing through the repository and also to create custom input forms. A huge disadvantage of this is that it only supports non-hierarchical metadata schemas by default.

2.1.5 Conclusions

None of the systems reviewed allows the creation of distributable collections besides Greenstone and even this requires prior installation of software, including a Web server before use. This justified the importance of a system like CALJAX which would create collections that would work straight off the storage medium used with the only requirement being a standard Web browser.

After considering the workings of the portions of these systems that dealt with management, it was observed that the main trend is to provide an abstraction to digital objects (such as “items” or “compound digital objects”) that might be affixed with metadata or contain the metadata within themselves. These basic units are then organised into collections on which the management operations are performed. Implementation solutions vary greatly between the different systems and within the systems themselves especially since they are highly customisable, e.g., they can be configured to make use of different file systems or database systems for storage.

Research into the existing digital repository systems was useful in producing a list of crucial features required of a digital repository management system, as well as a list of additional useful features. These are listed in Table 1.

Features which are required of digital repository software include accepting various standards of metadata, such as Dublin Core and MODS, and making data accessible, browsable and searchable through Web interfaces with good levels of usability. Creating items and modifying

items with different metadata formats are also necessary features. Allowing the extension of the system through plug-ins was also considered to be crucial. Configuration of the system should ideally be simple and minimal. Additional features include storing information about relations between digital objects, storing file formats in the hope of long-term preservation and providing security features such as access control for different items. Also, the conversion between different formats is a desirable feature of the management system since this would allow the interchange of objects between the different repositories.

2.2 Web 2.0 and AJAX Technologies

Web 2.0 is a broad term and is often considered to be just a "buzzword". Its validity is still being argued since most of its technologies are not new. The general tendency, however, is to consider Web 2.0 as a useful abstraction referring to a guiding set of trends and practices towards producing better applications (Maslov, Mikeal, & Legett, 2009).

Web 2.0 provides several features that enhance users' experience. AJAX (Asynchronous JavaScript and XML) technologies provide rich user interfaces that mimic familiar desktop applications (Coombs, 2007). Web 2.0 also deals with the interactivity between systems through sharing XML content. Web 2.0 features comprise a major part of the whole CALJAX system's interface and define the interactivity among its components.

AJAX is not so much a technology in itself as it is a group of interrelated technologies in widespread use, that are used together to produce highly interactive Web applications (Doernhoefer, 2006). AJAX incorporates (Garret, 2005):

- XHTML and CSS for presentation;
- Document Object Model (DOM) for dynamic display and interaction;
- XML and XSLT (Extensible Stylesheet Language Transformations) for data exchange and manipulation;
- XMLHttpRequest for asynchronous data retrieval and
- JavaScript, bringing everything together.

The main difference between the classical Web interface and AJAX based interfaces is the way data exchange takes place asynchronously (Garret, 2005). In the classical Web interface, each time a user chooses a hyperlink, a call to the server is made to request data from the server and this is transmitted back in the form of HTML. The user has to wait during the time the request is made and the data is retrieved from the Web server. Using AJAX, an engine is loaded initially and this engine can simultaneously render an interface for the user and communicate with the server. Therefore, whenever new data has to be retrieved, this does not stall user interaction. The data is normally transferred in the form of XML from XML or HTML servers. The main communication between the central repository and the browsing and searching parts of the distributable collections was done through the use of XML and XMLHttpRequest.

XML was developed by the XML Working Group, supported by the World Wide Web Consortium (W3C). Several design goals for XML made it appealing for use in CALJAX. These included being straightforwardly usable over the Internet, supporting a wide variety of documents, being easily processed by programs and also being human-legible (W3C, 1998). This made XML the logical choice for various purposes throughout CALJAX's management system. XML was used for:

- Storing configuration options for CALJAX's central repository management.
- Storing metadata for repository items.
- Creating browsable Web interfaces in conjunction with XSLT.
- Creating input interfaces for metadata using XML schema.
- Communicating updated item lists to distributable copies.

XSLT was used to enable the central repository to return XML results for all the listing operations but at the same time display them seamlessly as normal Web pages. XSLT enables HTML code to be produced from an XML server operation result by retrieving specific fields for use in the HTML code. This was used to create tables containing summaries of the XML list entries.

Metadata records vary widely in content but also in structure. For example, Simple Dublin Core is a linear set of fields while MODS 3.0 uses a complex hierarchical structure to store metadata (Network Development and MARC Standards Office of the Library of Congress, 2009). There is no standard method for inputting any general type of XML metadata and, generally, this is achieved through the use of custom plug-ins for each format in the system. XML Schema contain formal descriptions of XML records and are generally used for their validation. It has been suggested by Suleman (Suleman, 2003) that these can be used for producing general editors that work using XML Schema. The CALJAX management system made use of such a generalised editor for creating and editing metadata while also supporting custom editors.

DOM is a method of reading and changing HTML documents by traversing a hierarchical tree structure with node elements representing HTML elements (W3C, 2005). This was essential in the production of a dynamic Web interface, particularly with the highly interactive features supported, such as the metadata editors.

JavaScript was used for various purposes in the CALJAX management system. It was used throughout the interface to manipulate presentation elements as well as for more intensive tasks, such as generating XML from the various editors supported by CALJAX. One major limitation of the JavaScript language is its lack of file access functions. The main reason for the omission of this feature is to prevent security violations on the client's machine (Flanagan, 2002). This meant that JavaScript had to be generated dynamically in the same way as Web pages with parts of information hard coded. Another issue with the use of JavaScript was that the various different

Internet browsers interpret the same JavaScript code differently and careful treatment and testing was required so that the JavaScript was compatible with all the major browsers.

Chapter 3

Design

This section discusses the scope and limitations of this project and then details the design solutions for the various components of the system during its development.

3.1 Scope and Limitations

Ideally, the system would be able to cater for all the needs of such a central repository management system. However, since this project aims to simply demonstrate the feasibility of such a system and because of time constraints, the main focus was on implementing the most critical features.

3.1.1 Browsing and Searching Functions

Both of these functions are standard features of a digital repository management system. Even though the other parts of CALJAX dealt with browsing and searching through the collection, these worked mainly on pre-processed copies of the repository. To implement live search functions, inverted files would have to be created but also maintained as changes in the repository take place. This is beyond the scope of this project. Browsing also generally would allow sorting on metadata fields. Such an operation would necessitate the implementation of index files and is also beyond the scope of this project.

The management system does implement simple methods of browsing and searching and these are discussed below.

3.1.2 XML Schema Editors

To support various types of metadata, CALJAX makes use of XML Schema files. While they do read the whole structure of the XML Schema and produce a tree hierarchy of all the possible field values, the editors do not read in validation information from the schema files. XML Schema files contain information such as enumerations of possible values for specific fields, field formats, minimum and maximum number of occurrences for each field and also annotations with comments on the fields. This information could be potentially used to facilitate and validate user input but this was not implemented due to time constraints.

3.1.3 Retrieval of Updated Items

In the current system, the distributable collections can request a list of items that have been updated from the central repository. For simplicity, it was assumed that this list of items would be sufficiently small to be practical. This is a reasonable assumption since only metadata files are sent. However, a practical system might be required to only retrieve a few of the updated items as they are required. This would also have required an elaborate search scheme on the central repository. Since this would have taken up considerable development time, and was not considered a critical portion of proving the feasibility of the system, a simple list of updated items was used.

3.1.4 Security and Access Control Features

Security evaluations would need to be carried out for such a system to prevent any loss of collection data. Access control might also be required to choose who can edit which repository items. These were not implemented due to time constraints.

3.2 CALJAX System Overview

Figure 1.1 shows how the different components of CALJAX interact with one another. The collection items are stored in structured directories in the file system of the central repository system. These can only be modified by server side software. The central repository system communicates with the components of the system mostly through standard HyperText Transfer Protocol (HTTP). This allows requests to be made for carrying out actions such as the management of the repository and retrieving update item lists.

However, the browsing and searching back-ends of the system directly access the file system on which the collections are stored. This direct access is justified since the browsing and searching functions of CALJAX require intensive pre-processing to produce specific indices for their purposes and direct access is much faster than HTTP communications. In order for this to work, a standard structure for the repository was agreed on. This is described in more detail in Section 3.4.

3.3 Management System Overview

Reviewing several existing repository systems was useful in producing a list of features expected of the management system. Table 1 shows a list of these features along with their respective importance with respect this project. All the features listed as “Crucial” had to be implemented in order to demonstrate the feasibility of the system. As many of the other features as possible were to be implemented depending on time constraints.

Figure 3.2 shows the structure of the system that was implemented.

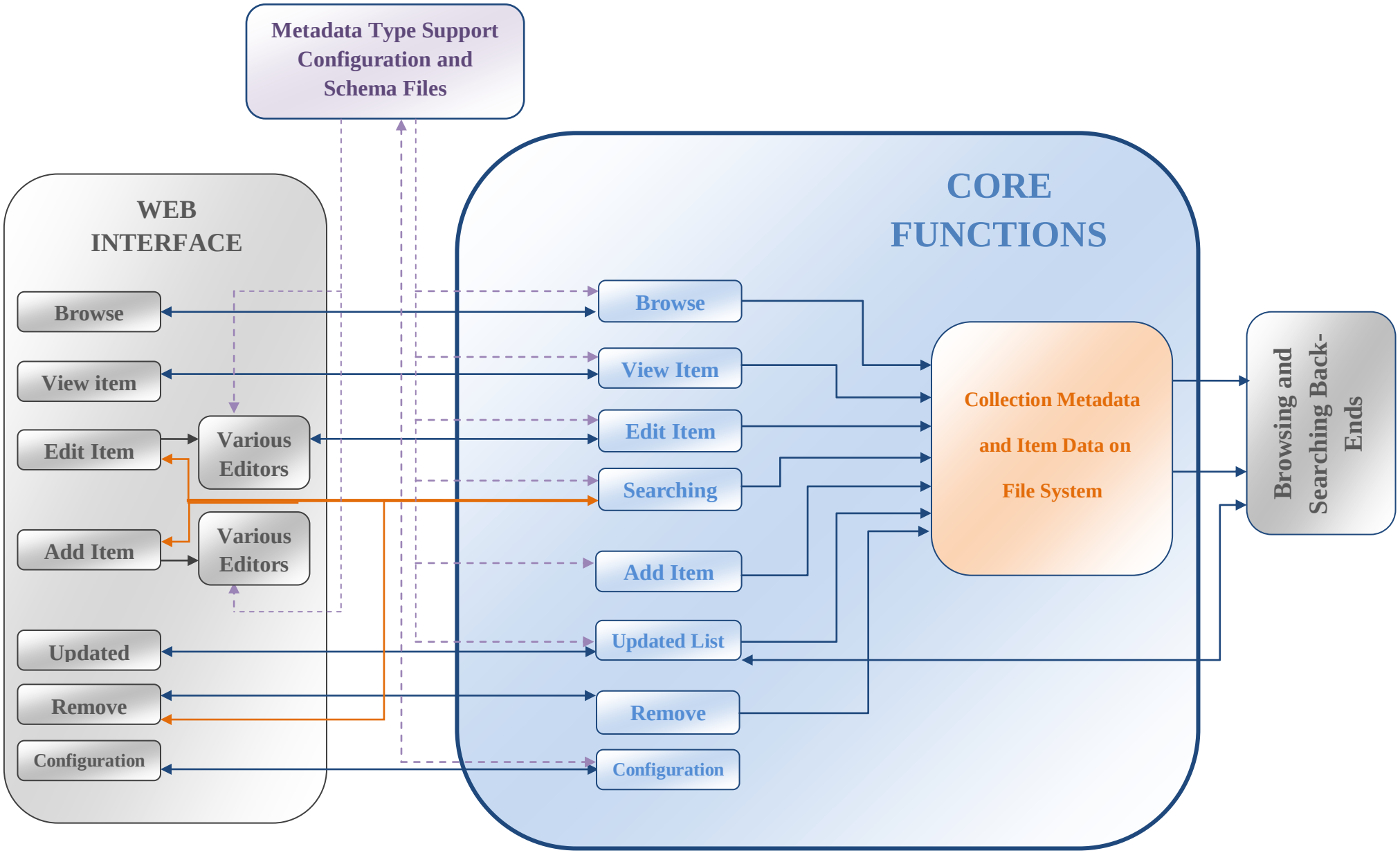

Figure 3.2: Overview of the structure of the management system of CALJAX.

The management system comprises of the collections stored in the file system on the server in a specified structure. These are accessed by the management's server which provides a set of core functions to be carried out on the collections.

Feature	Importance	Implemented
Support for various metadata standards	Crucial	Yes
Creating repository items	Crucial	Yes
Modifying repository items	Crucial	Yes
Browsing through a repository	Crucial	Yes
Searching the repository	Crucial	Yes
Web interface with good usability	Crucial	Yes
Providing list of updates	Crucial	Yes
Custom plug-in system for supporting new types of metadata	Important	Yes
Viewing repository items' metadata	Important	Yes
Scalable to large collections	Important	Yes
Simple configuration	Desirable	Yes
Validation and integrity of data	Desirable	Partially
Relations between digital objects	Desirable	Partially
Conversion between different formats	Low	No
Storing file formats for long-term preservation	Low	No
Access control	Low	No

Table 1: Features of the CALJAX digital repository management system.

The core functions take in HTTP parameters and carry out the operations on the collection. For example, Add Item takes in a parameter containing the metadata of the item to be uploaded and another containing the metadata type, amongst others. The Web interface builds on these core functions, by taking in input parameters and converting the results of the server operations to presentable forms.

The separation of the core functions from the Web interface was important for the provision of customisable interfaces, particularly in the cases of adding and editing repository items. This structure greatly simplifies the creation of plug-ins which do not have to manipulate the metadata collection directly. Instead, these plug-ins can be developed based on a specification of the core functions' parameters only.

3.4 Repository Structure

The repository structure had to be decided early on during the development of the system as all the three main components of CALJAX accessed the repository directly. Several existing repository systems store their collections using database systems and others allow customisations of the file system used. For the purpose of this project, a simple specification was sufficient and advantageous.

The manipulation of data is done using Java libraries so CALJAX does work on various file systems, depending on the Java platform used. A database system was considered unnecessary because it would increase the complexity of the system with benefits not essential to the goals of this project. Also, a database system would imply further software requirements and would possibly reduce the number of platforms supported by CALJAX.

The items are stored in a repository directory and they can be classified into subdirectories. An item is defined as a set of two files – a binary file containing the item's data and an XML formatted file containing the metadata for the item. All items follow the naming convention whereby the metadata file has ".metadata" affixed to the item's binary file name. Any file with a corresponding file having the same name with ".metadata" affixed to the end of its name is considered to be a repository item. Files with no such corresponding metadata files are ignored. The metadata file has to be in the same directory as the binary file. The item's name is defined to be the binary file name including the directory path relative to the repository directory. This simple specification was chosen because it provided a simple but unambiguous representation of a collection.

Each metadata file has an XML root element appended to their metadata XML contents. This is used as a header containing information such as the namespace definitions relevant to the metadata and a signature indicating the type of metadata it contains. This header is particularly important for XSLT transformations of the XML files into presentable HTML formats, where namespace specifications have to be strictly respected.

3.5 Web Interface Design

A Web interface was chosen for managing the system because this would offer a platform-independent means of doing so. The Web interface implemented had to work on all the major Internet browsers. This would facilitate the remote management of a central repository server.

User interface design was an iterative process, highly dependent on user evaluations and compatibility evaluations of the system. These evaluations are described in detail in Chapter 5.

3.5.1 Version 1

The first functional version of the system featured mainly the implementation of the system's core functionality. This meant that, even though it featured a Web interface, this was simply a means of communicating with the server to test the core functionality. It consisted of buttons for each core function and simple input fields for the various parameters.

No user evaluation was carried out on this system because it was not expected that a working version of the Web interface would bear any resemblance to it.

3.5.2 Version 2

Version 2 was the first evaluated version of the system. It featured implementations of all parts of the Web interface besides item editing. The Web interface was broken down into a navigation menu and embedded iframes that provided a means of seamlessly switching between the various management options. Since the server gave results of queries as XML listings, this version made use of XSLT to produce tables for lists. The tables were built based on key field specifications for each format. To provide consistency and more aesthetic appeal, version 2 made use of Cascading Style Sheets (CSS) throughout the Web interface. It featured various types of editors, including a custom Dublin Core editor, a plain XML editor and a functional version of the XML Schema based editor. These editors made use of JavaScript and DOM for producing interactive pages that resemble standard applications.

3.5.3 Version 3

Usability testing and compatibility testing of version 2 revealed several major issues in its design. An attempt was made to fix these problems in version 3 and version 3 underwent the most important usability testing out of all the versions of this software.

3.5.4 Post Version 3

The rigorous usability testing of version 3 pointed out very specific problems with the Web interface and most of these were corrected to give the final system. This system was evaluated and found to meet the initial goal of producing a Web interface for the management system with good usability.

3.5.5 Results of Intermediate Usability Testing

3.5.5.1 Browsing

Even though the changes made to the browsing section of the interface were minor as a result of usability testing, they fixed a major design problem with the system. In version 2, the system used XSLT to parse the metadata XML that sent as a result of the listing operation. This produced tables listing the key fields for each type of metadata as specified in the configuration file. There was no indication of the unique item name (defined in section 3.4) of each item. The evaluator of version 2 assumed that the "Title" field (found in both metadata types used in that version of the system) was the identifier and subsequently typed the value of these fields in as item name. All repository item listings were changed to make it clear what the item name was.

Another change was the use of JavaScript and DOM to remove any empty tables that only contained metadata field headers and no items.

3.5.5.2 Viewing Items

As shown in Figure 3.3, viewing an item in version 2 involved typing in the name of the item following which the user would be presented with a listing of the metadata and a preview of the resource item if it was an image. However, the evaluator typed in metadata content instead of the

item name. As a result of usability testing, the system now provides a few options to help users find the item they are looking for. This includes the option of searching through the repository, browsing through the repository or directly typing the item name.

Figure 3.3: Viewing an item in version 2, before any usability evaluation was conducted.

As shown in Figure 3.4, in post version 3 it is made clear to the users that there are several options available to them, by differentiating the sections. Also, information about how the search only applies to item names is made prominent to avoid mistakes by users. If browsing or searching is chosen, users are provided with a listing of the repository matches, from which they can select the item they were looking for.

3.5.5.3 Retrieving List of Updated Items

The changes to this section are the same as the changes implemented for browsing (Section 3.5.5.1).

3.5.5.4 Adding Repository Item

For the addition of items, users can choose between the various available metadata editors for the metadata type they want to use for the item. All the editors for each metadata type are listed depending on the specifications in the metadata configuration file.

Figure 3.5 shows the use of the Schema based editor to add a new repository item in version 2. This figure is the size of the viewing panel in the browser. It shows that a lot of viewport space was wasted at the bottom of the screen and between the menu on the left and the editor (green arrow). Subsequent user testing showed that it was not clear that, for each metadata format, different editors were available. The flow of usage was also confusing to the user as illustrated by the arrows. The section containing the "Generate Metadata" button and the XML contents were centred vertically in the editor. This means that as metadata got more complex, users had to scroll to the centre of the editor to look for it.

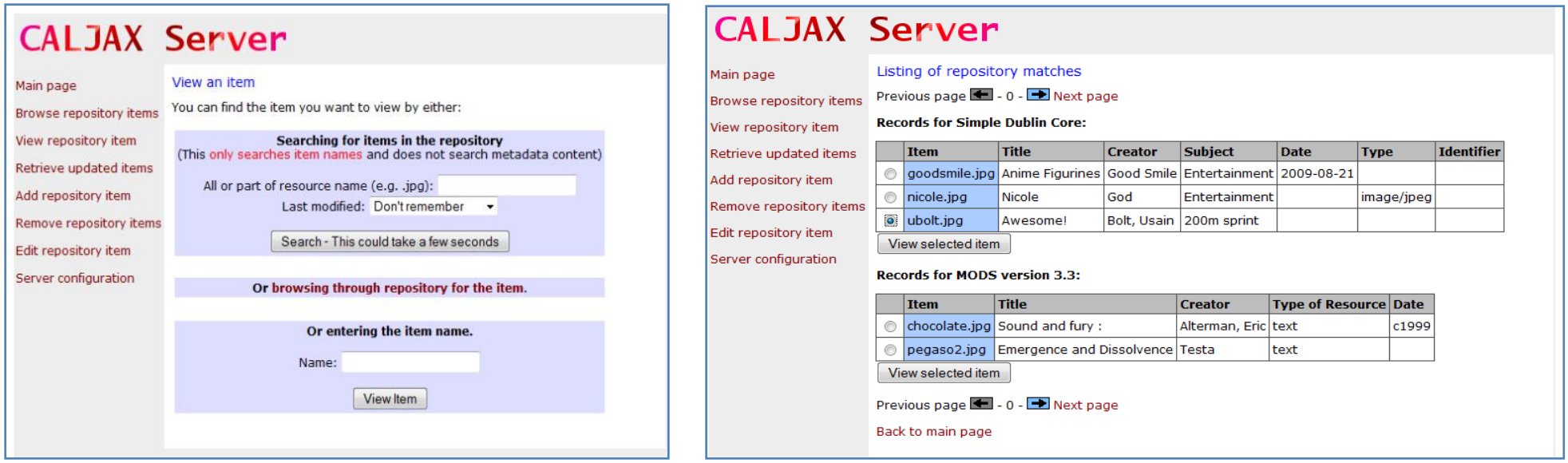

Figure 3.4: Screenshots showing the two steps for viewing an item in post version 3.

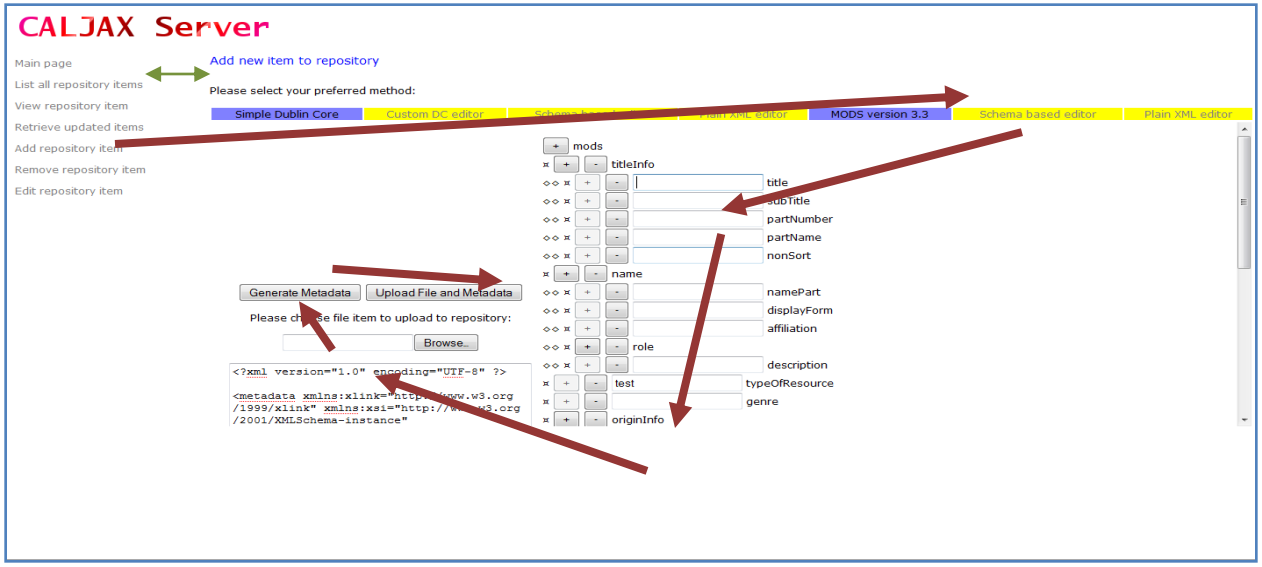

Figure 3.5: Screenshot of "Add repository item" function before any usability evaluation was conducted.

Figure 3.6 is an illustration of how the iterative process of evaluation was effective. The flow of usage has been made much more natural, as shown by the arrows. Also, the window dynamically changes size to maximise the proportion of the viewing panel in the browser used by the editor. This applies to the whole Web interface but is particularly important in the metadata editors where screen space is important for complex metadata. The spacing between the menu on the left and the editor has been minimised too. This has been made possible by shading the background of the menu.

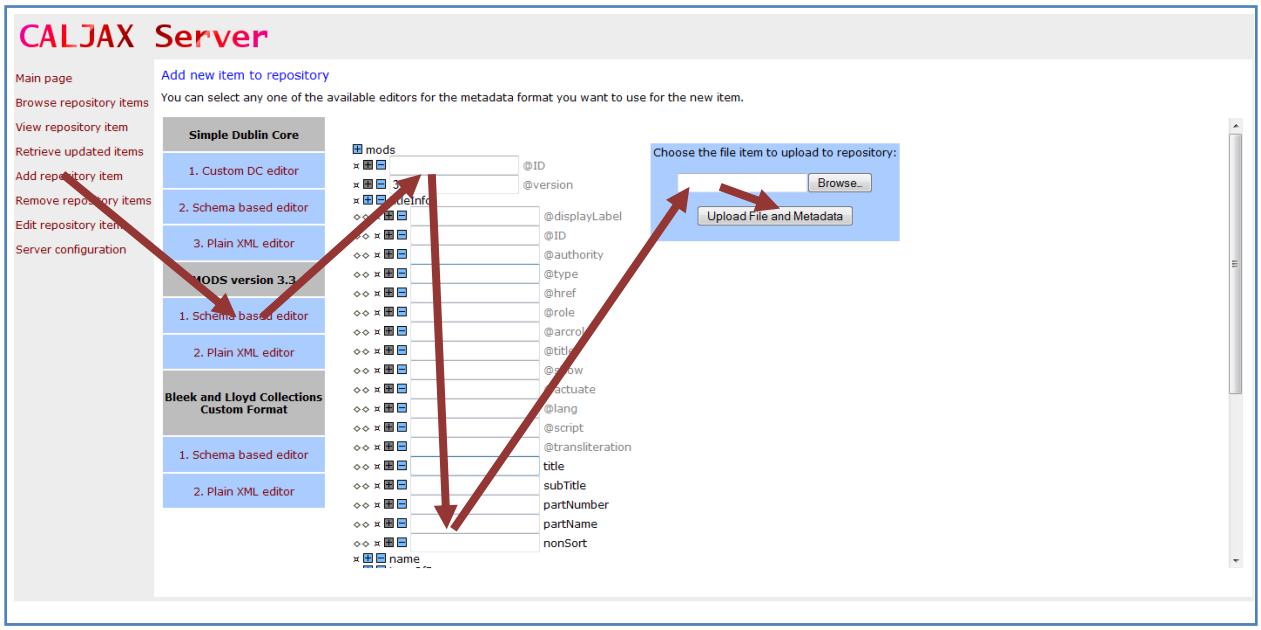

Figure 3.6: Screenshot of "Add repository item" function after last usability evaluation.

The "+" and "-" buttons have been changed from buttons to images so that they do not look different on different browsers. The explanatory text has been elaborated to assist any confused users. The format headers have been made bold and the editors have been enumerated to make it clear that they fall under the headers. The "Generate XML" button has been completely removed and this is done automatically. The box containing the "Upload File and Metadata" button has been highlighted and is always at the top of the screen and only moves horizontally as the metadata gets more complex.

It should be noted that the implementation of a new feature has provided a new challenge for the interface design. Displaying attributes as well as normal fields for the metadata means that it is a harder for users to locate any particular field. This could not be tackled due to time constraints.

3.5.5.5 Removing Repository Item

Removing an item involved typing the item's name and this would produce a preview of the item asking whether to confirm the deletion of the item. This was changed in version 3 to allow users to browse or search for the item they wanted to delete. Also, users can delete multiple items from

the collection by making use of checkboxes in the list of items provided. This produces a preview of all the items selected for deletion before the operation can be confirmed or cancelled.

3.5.5.6 Editing Repository Item

Version 2 did not feature any editing function. Version 3, however, combined the usability design of adding items and viewing items to produce the editing user interface. This is because the editors used in creating repository items are similar to those used in editing the items.

Users are provided with the options of searching or browsing for the item they want to edit or they can directly specify the item name. Once an item has been chosen, the system uses the metadata type signature in the header of the metadata file to produce a choice of editors for that particular metadata type to the user. Depending on the editor chosen, the fields are automatically filled with values read in from the metadata file. The users can choose to overwrite the binary data file of the item or can opt to leave it unchanged and only edit the metadata.

3.5.5.7 Server Configuration

Initially, it was expected that server configuration would take place through direct editing of the configuration files. However, the switch to a single XML configuration file in version 3 made it easy to provide a user interface similar to the XML Schema based editors for editing the server configuration.

3.6 System Features

3.6.1 Browsing Repository

Since the repository contained items of various types and the core listing functions from the server returned an XML formatted list of results, XSLT (Extensible StyleSheet Language Transformations) was the logical choice for producing a user friendly interface.

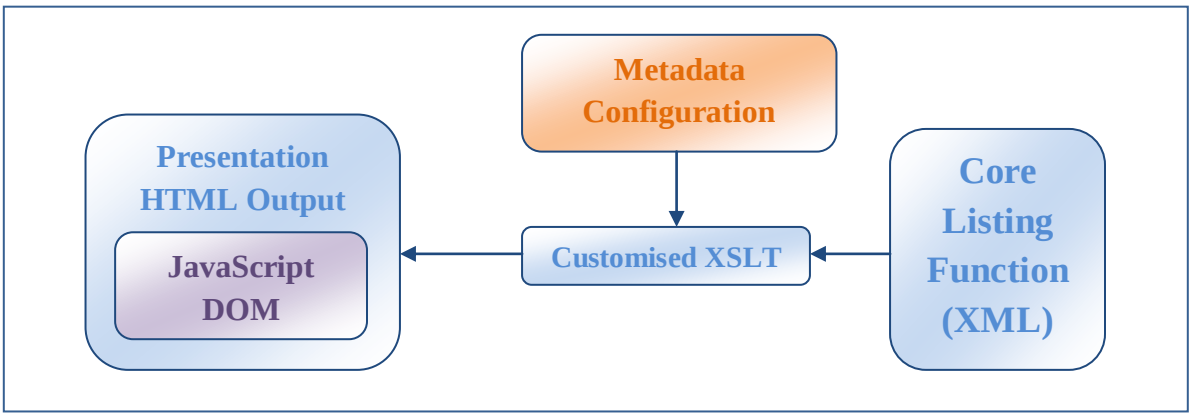

Figure 3.7: Formatting core server output.

The XSLT is generated dynamically depending on the metadata configuration of the system. This is shown in Figure 3.7. The metadata configuration file contains the various namespace

definitions for each metadata format. As mentioned in Section 3.4, all the metadata files have a root element containing namespace definitions as well as other header information such as the item name. The XSLT creates a table for each metadata type and uses the headers specified in the metadata configuration file for each metadata type. The root element of each metadata file has the same name for all items but it is found in the default (and unique) namespace of the metadata type of the item (Section 4.3.1 contains an example of a metadata list returned by the server). Therefore, by making each table list only those items having a root element in the default namespace, the items are sorted into their respective tables.

The drawback of this method is that it is not possible to count the number of items in each table and therefore empty tables' headers still appear. This was fixed using JavaScript DOM manipulations to detect and remove any empty tables in the HTML output.

The same solution is used for selecting items to edit, view or delete except that the XSLT generates HTML forms that return the chosen item names and any other information such as metadata signature to the corresponding core module.

The browsing method implemented does not allow for sorting of metadata on the various fields since this is beyond the scope of this project. The browsing implemented for the distributable copies is more flexible but was not aimed at working on the central repository.

3.6.2 Searching Repository

The basic search function implemented allows users to specify part of the file name to be searched for and choose from a range of dates from which the item was last modified. The search function works by recursing through the directory structure of the repository and finding valid items. These are then compared to the search string and the file timestamp is matched against the modification range provided. As soon as the search has the number of items to fit into a page, it returns.

A proper search function would have required prior indexing of the collection's metadata and would also have to maintain these indices as metadata was changed. It should be noted that the search function implemented for the distributable copies of the system was not aimed at searching the central repository. Such a search function for the central repository is beyond the scope of this project.

3.6.3 Viewing Collection Items

It was noted that most browsers featured their own XML tree viewers. Therefore, since writing a custom metadata viewer would take a significant amount of time, these features would be exploited. When viewing an item, users are presented with a window containing the XML tree of the metadata and a preview of the item if it is an image. Users can also follow a link to directly view or download any item from the repository over an HTTP protocol.

3.6.4 Removing Collection Items

In the initial version, choosing to remove an item resulted in both the content file and the metadata file being permanently deleted from the repository directory. This was deemed to be an unsafe action so the procedure was changed to a safer alternative.

Removing items is achieved by renaming the metadata file of the item and its content file to new files with a ".old" extension. If any such files exist, they are overwritten. The purpose of this is to enable the restoration of accidentally deleted files but also to invalidate the item. By the naming convention's definition of an item defined in Section 3.4, the two files no longer represent an item and are no longer listed in the repository even though they have not been erased permanently.

3.6.5 Adding and Editing Collection Items

Users are presented with a list of possible editors. In the case of editing, this is limited to the editors for the metadata type of the item chosen. After the user chooses an editor, users are presented with an interactive HTML form through which they can enter the metadata. For editing, this form is pre-filled with values read in from the metadata file.

3.6.5.1 Custom Editors

In the metadata configuration, for each metadata type, custom editors for adding and editing repository items can be chosen. This is described in more details in Section 3.8.

Most existing repository management systems use plug-ins to edit the different types of metadata. Custom editors are more user-friendly and can carry out more validation checks than generalised metadata editors.

A custom editor for the Simple Dublin Core format was built to test the extension system but this could not be done for the MODS 3.0 format (Network Development and MARC Standards Office of the Library of Congress, 2009) used because of its complexity and time constraints on the project.

3.6.5.2 XML Schema Based Metadata Editing

Even though custom editors are bound to provide better interfaces, these custom editors need to be developed and tested in a time consuming process before they can be used. For this reason, an XML Schema based metadata editor was built to allow the system to support different metadata formats.

Figure 3.8 shows the structure of the XML Schema based editor. The server-side module reads in the XML Schema file as specified in the metadata configuration file. This is used in combination with the contents of the metadata file being edited to generate a structure based on the Schema structure and organised into hierarchical SPAN sections (a SPAN is an HTML element that can be used to subdivide an HTML DOM structure into sections that can be manipulated

individually). XML Schema are organised into groups or complex types which can contain combinations of groups, complex types, simple types or primitive types. The complex types are broken down till simple types or primitive types are encountered and input fields are provided for these. Attributes are also taken into consideration when reading the XML Schema structure.

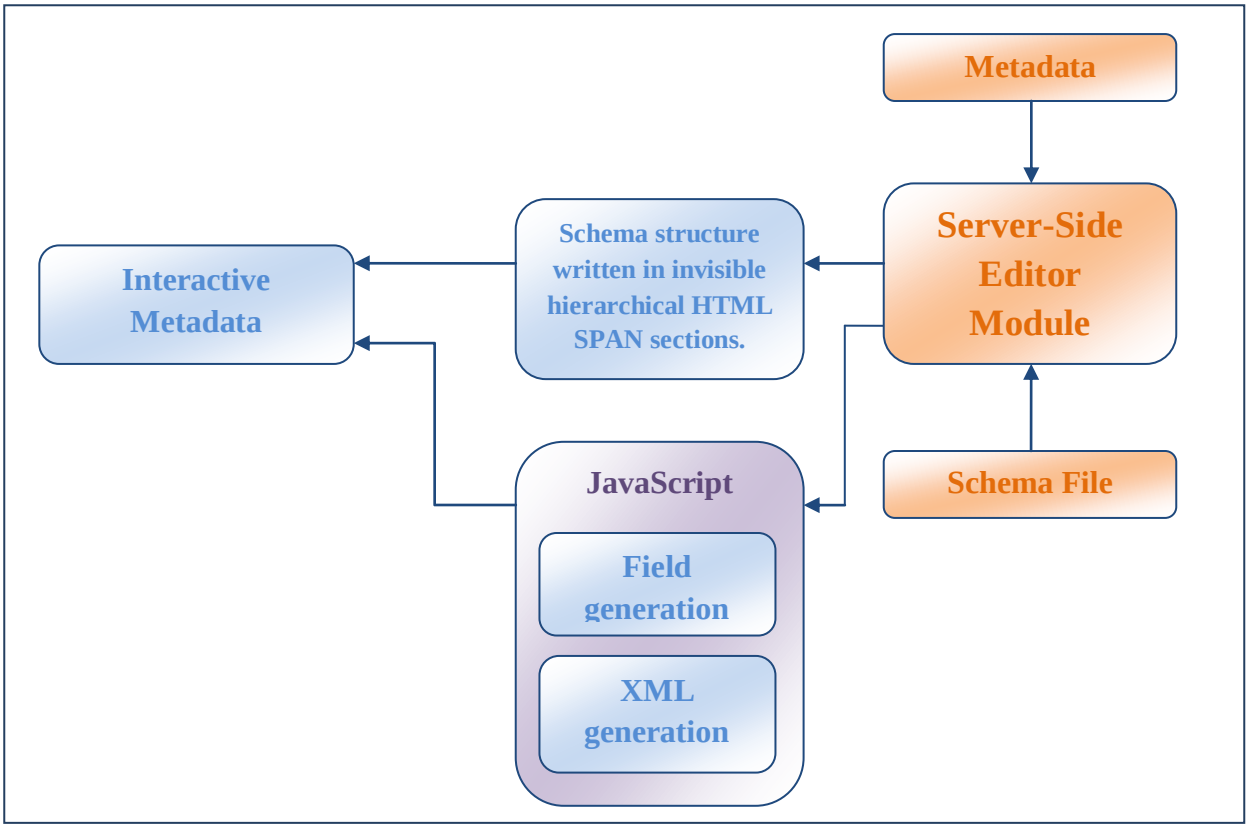

Figure 3.8: Schema Based Metadata Editor.

The field generation portion of the JavaScript editor is responsible for finding the correct part of the hidden structure when a field is expanded and copying it to the correct position on the form. An important characteristic of XML is that it is developed in a way to be human legible (W3C, 1998) and the field names used are generally descriptive. The XML generation reads through the current visible HTML structure, which is also organised hierarchically through SPAN sections and, in this way, is able to generate correct XML code recursively.

This design was preferred over a model that would make server requests every time a new field was generated because it was more efficient communication-wise. Also it maintained the consistency of the system design as the client side processing enabled the use of the simple core functions for adding and editing items and did not require new core functions to be developed.

In version 3, a drawback of this system was noticed when using extremely complex metadata formats such as MODS 3.0 as MODS 3.0 records can contain other MODS 3.0 records and are therefore recursive. While the schema-based editor limited recursion by setting the depth of the

tree explored when creating the HTML Schema structure, the size of the structure was still significant despite several minor modifications to reduce the size.

Interface Web Page	Size (KB)
Browse repository items	8.6
Custom Dublin Core editor	8.0
Dublin Core Schema based editor	7.3
MODS Schema based editor	604.8

Table 2: Interface Web page sizes (excluding associated images and script files) in version 3.

As Table 2 shows, the MODS Schema based editor was much larger than the standard Web page sizes. Dublin Core, which is a relatively simple metadata format with a flat structure layout, produced a reasonably-sized Schema based editor.

In post version 3, the schema-based editor was modified to be able to detect any duplication of structure and store a single copy. This required further JavaScript DOM processing since a single copy meant that copies had the same indentation independently of their depth. This change also brought the advantage of no limits to the depth of field expansion. The final page size was reduced to 215 KB, which can be considered to be a reasonable size given the complexity of the MODS structure. These changes were only applied to the Schema based editor for adding new items but could not be applied to editing items due to time constraints.

3.6.5.3 Plain XML Editor

A plain XML editor was also provided for advanced users of the system. Since the core system does not allow for invalid metadata to be submitted, any invalid XML typed is rejected. The editor automatically fills in the XML header namespace definitions depending on the metadata type chosen.

3.7 Scalability Design

The scalability of the system was tested by applying the system to a large collection of items. By the end of version 2 of the software, the collection of items that was used contained less than 20 items with different types of metadata. A larger collection of items was needed to test how the system could deal with real world collections.

For this purpose, the Bleek and LLoyd Collection was chosen. This is a collection of notebooks and drawings about the Bushman groups in South Africa. This data was collected mainly by Wilhelm Bleek and Lucy Lloyd in the late 1800s. A sample of around 15,000 images was chosen to test the system – the equivalent of around 2.7 gigabytes of information.

Before the sample collection could be tested, it had to be converted to a format that was supported by CALJAX. All the collection information was contained in a single large XML file. This had to be broken down in such a way as to retrieve all the information possible about each individual image and create one XML metadata file for each image in the collection. This

resulted in a significant duplication of information and the metadata increasing from around 5 megabytes to around 100 megabytes.

When this was used in CALJAX, the browser became unstable and crashed as the lists retrieved were too large to be handled and parsed. For version 3, therefore, the listing system had to be re-written to break down results into separate pages. This also had several repercussions on the system features because their usability was compromised when browsing through such a large collection and therefore a search feature was also required.

User testing showed that users were satisfied with the response time of the system (Table 7 and Table 9). Even though the search system implemented is very basic, processing only filenames and simply recursively going through the list of all files, this process returns results, in the worse cases (e.g., when no file matches are found, so the system has to scan through the whole repository), in less than 3 seconds (Section 5.4.3). Most of the lists are returned immediately as the system only searches for a number of matches for a single page. It is observed that the first worst case search on the server can take up to 15 seconds. However, the underlying operating system caches the file lists and therefore subsequent searches are much faster. Since it is expected that a server host machine is running most of the time, users will never encounter these significant search times.

3.8 System Extensions and Configuration File Structure

The metadata type configuration file is stored in a single XML file. Each metadata type has a name, a string signature, namespace definitions and can contain links to any custom editors. The key fields used in repository listings are also specified.

Custom editors are in the form of Java Server Pages which are interpreted by the server. These need to produce XML code from their input and submit it to the core functions for editing and storing items using the HTTP parameter specifications described in Section 4.3.2. This provides a simple but flexible way of extending the system's support of various metadata types.

Chapter 4

Implementation

This section provides details about the implementation of CALJAX management system. A specification of the development platforms used is provided followed by details of the iterative approach to implementing the system. It also provides specifications of the communication protocols used.

4.1 Platform and Language

The management system of CALJAX was developed using Java Server Pages (JSP) technology. This was chosen mainly because it offered a platform independent solution to a Web interface. JSP proved useful in mixing HTML generation with Java code. It allowed extension libraries intended for the Java language to be used in writing the server. Another advantage of using Java Server Pages was the ease of deploying such a system. The only requirements were the installation of a Java runtime environment and the Apache Tomcat server. The system was developed and tested under Java Runtime version 1.6.0_13 using Apache Tomcat 6.0.18 under Windows XP version 5.1. JSP was also critical in the extension system used as it provided a powerful but very simple means of extending the system.

The XMLBeans library was used to read the XML Schema structures.

4.2 Iterative Design

Since the management system consisted of various components, an iterative approach was chosen. This meant that at all stages of development there was a working prototype. The core functionality was implemented first following which additional features were added or existing features were improved based on the results of the various types of evaluation conducted.

It should be noted that a major portion of development was spent improving components already implemented rather than implementing new features.

4.2.1 Iteration 1: Version 1 (21/08/2009)

Version 1 provided basic features and was more of a pure server interface testing the core functions than a usable system.

- Listing all items.
  - Produces an HTML list of all items in repository directory.
- Listing updated items.
  - Produces an HTML list of updated items.
- Viewing an item.
  - View item and metadata for an item with specified name.
- Add new file (only Dublin Core).
  - Could create multiple "Creator" fields.
    - This was done through new server requests and had to download the whole form each time a new field was requested.
  - File upload (processing of multipart HTML header data).
- Edit file metadata.
  - Could read Dublin Core file but it would only allow multiple "Creator" fields.
  - Could not create more fields than were already in "Creator" fields.
- Delete file metadata.
  - Deleted file with a specified name.

4.2.2 Iteration 2: Version 2 (06/10/2009)

This was the first functional version of the software and the first version to undergo evaluations.

- Listing of items.
  - Using XSLT formatted XML.
  - XSLT generated in such a way as to produce tables containing specified fields from each format.
- Listing updated items.
  - Same as for listing of items changed to XML formatted with XSLT.
- Viewing item.
  - Unchanged.
- Multiple metadata types supported.
  - Configuration files.
    - Specification of custom editors.
    - Schema file locations for Schema-based editor.
    - Namespace definitions.
    - Listing field specifications.
- User interface.
  - Menu providing overview of system and iframe for navigation.
  - Stylesheets for HTML pages (and translated XML pages).

- Adding new items.
    - Support for multiple metadata formats.
    - Custom editors can be configured.
    - Created custom Dublin Core editor.
        - Supported multiple copies of any field using JavaScript. (No server interactions).
        - XML code can be generated and then submitted.
    - Schema based editor.
        - Could read an XML Schema and produce an editor.
            - Did not feature XML attribute fields.
        - XML code can be generated and then submitted.
        - Uses JavaScript, DOM and XMLBeans to achieve this.
- Editing.
    - No more functional (Dublin Core editor was obsolete).

4.2.3 Iteration 3: Version 3 (22/10/2009)

Version 3 was a result of fixing critical issues with version 2 and general improvements following its evaluation but also featured several new components. It was a functional and usable system.

- Scalability and Performance.
    - Tested on and fixed to support large collection of items.
- Compatibility.
    - Various changes in presentation and JavaScript for cross-browser compatibility (Internet Explorer, Mozilla Firefox, Opera, Apple Safari).
- New metadata format supported: Bleek and Lloyd Collection Custom Format.
    - Created Schema file for this format.
- Directory structure of repositories.
    - Allowed sub-directories.
- User Interface.
    - Major changes to user interface described under the different function headers below.
    - Automatic resizing of iframes depending on window size.
- Listing of items.
    - Broke down results into browsable pages. Dynamic XSLT created tables listing results depending on configuration.
    - JavaScript in XSLT formatted XML uses DOM to hide empty tables.
- Listing of updated items.
    - Drop-down menus to select date.
    - Broke down into pages which could be browsed.

    - The update function used by distributable copy interfaces was still available.
- Viewing items.
    - Search function and browsing function to find file to view.
- Adding item.
    - Interface changed to provide more natural flow of usage (used columns).
    - Automatic XML generation simplified the interface.
    - Schema-based editor.
        - Fixed problems with inputting field data that invalidated XML (e.g., characters like "&" are automatically converted to """).
        - Several modifications to reduce size of editor.
    - Validation checks on input XML before storing to repository (not schema based).
- Editing item.
    - Search and browsing options to look for file to edit.
    - Supports various metadata types.
    - Can be configured for use of custom editors.
    - Automatic XML generation.
    - Schema-based editor could read in XML metadata from any file in the repository.
        - Uses JavaScript, DOM of HTML editor form and XMLBeans as well as DOM of metadata instance XML.
        - Input field data is validated (fixes invalid characters).
        - Several modifications were made to reduce size of editor.
    - Validation checks on edited XML (not schema based).
- Deleting item.
    - Search and browsing options to look for file to delete.
    - Multiple file deletion.
- Configuration File.
    - Converted to XML.
    - Used schema based editor to produce a quick Web interface editing tool for the server configuration.

4.2.4 Iteration 4: Post Version 3 (29/10/2009)

Further refined system based on evaluation of third version as time constraints allowed.

- User Interface.
    - For add and edit repository item functions, the metadata formats were written in bold and the editors were numerated to make it clear that these were available as options. This was the most common usability issue.
    - Error messages throughout system fixed to be more helpful and meaningful.
- Adding Item.
    - Schema-based editor.
        - Support for XML attributes in metadata.

        - Schema-based editor was re-written to reduce size dramatically.
        - No more limiting of infinitely recursive schemas through depth (linking similar schema types instead of storing them multiple times).
- Editing Item.
    - Schema-based editor.
        - Support for XML attributes in metadata.

4.3 Communication Protocols

4.3.1 Retrieving List of Updated Items

A standard HTTP 1.1 protocol request should be made using either GET or POST parameters to:

http://[server address]/ListUpdatedServlet

The parameters are shown in Table 3.

Parameter	Comments
time	In the format XX:XX (e.g., 23:59).
day	Integer from 1 to 31 specifying day of the week.
month	Integer from 1 to 12 specifying the month of the year.
year	Integer specifying a year.

Table 3: Parameters for retrieving list of updated items

Returns XML file with root element <metlist> containing each item's metadata in a <metadata> element if successful. The <metadata> element contains the namespace definitions for the metadata. For example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl"
href="XSLgeneratorListUpdated.xsl?page=0&amp;day=5&amp;month=11&amp;year=2009&amp;time=22:21
"?>
<metlist>
       <metadata xmlns="http://www.loc.gov/mods/v3"
       xmlns:xlink="http://www.w3.org/1999/xlink"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" itemID="CREDITS.html"
       signature="mods33">
           <mods version="3.0"><titleInfo><title>Credits</title></titleInfo></mods>
       </metadata>

       <metadata xmlns="http://purl.org/dc/elements/1.1/"
       xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:dcterms="http://purl.org/dc/terms/"
       itemID="LICENSE.txt" signature="simpledc">
           <title>License</title>
           <creator>Suraj Subrun</creator>
       </metadata>
</metlist>
```

In case of error, the XML file will contain the same <metlist> root element but it will contain an <error> element with the error description. For example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl"
href="XSLgeneratorListUpdated.xsl?page=0&amp;day=5&amp;month=11
&amp;year=2009&amp;time=2221"?>
<metlist>
        <error>Invalid time specified.</error>
</metlist>
```

4.3.2 Custom Editors for Adding New Item and Editing New Item

The editor should be a valid JSP file whose location is specified in the metadata configuration file.

The standard HTTP 1.1 protocol is used for communication.

The input parameters listed in Table 4 must be read from both GET and POST requests:

Parameter	Description
Id	Integer specifying type of metadata type.
Item	(For edit item only) Item name of item to be edited.

Table 4: Parameters read in by custom editors

The output parameters listed in Table 5 should only be encoded using POST multi-part form data and these should be submitted to:

http://[server address]/AddItemServlet for adding an item.

http://[server address]/EditItemServlet for editing an item.

Parameter	Description
xmlContent	XML formatted metadata. XML should have a root called <metadata>. For edit only, this root must have an attribute named "signature" containing the metadata type's signature and another named "itemID" containing the item name.
schemasignature	(For add item only) Metadata type's signature (should be same as in metadata configuration file).
Item	(For edit item only) Item name of item to be edited.
file	Must be of "file" type. Item content data. This value should not be specified to leave item in repository unchanged (for edit item only).

Table 5: Parameters submitted by custom editors

The server will return an HTML page describing the result of the operation.

Chapter 5

Evaluation

This section outlines the results of the various methods of evaluating the system. Evaluation was carried out at the various iterations of the system. User testing was carried out to find usability issues with the system. The other types of evaluation conducted were performance testing and compatibility testing across a range of different Internet browsers.

5.1 Evaluation of Version 1

Since version 1 of the system was a bare-bones layout of the system's basic functionality, no evaluation was performed.

5.2 Evaluation of Version 2

After version 2 of the system, some minor user testing was conducted but even this pointed out major flaws in the design of the system. Compatibility testing across the major Internet browsers also produced several major flaws with the system as several system functions did not work at all on some platforms. Scalability testing also pointed out major problems with the systems' handling of large amounts of information as described in Section 3.7.

The evaluation of version 2 of the system proved extremely useful in producing a significant list of changes that were needed for version 3. Several additional features that were initially planned to be implemented in version 3 had to be dropped in favour of fixing the issues brought out by the evaluation of the second version of the system.

5.2.1 Usability Testing

User testing was carried out by having a single evaluator carry out tasks on the various components of the system.

A major problem with the system was that listings of repository items did not contain the repository item names. This meant that when prompted to enter an item name, such as when

selecting to view an item, the evaluator used the "Title" field from the metadata content unsuccessfully.

The evaluator also found the choice of editors while adding a new item to be confusing and also the order of usage of the editors was confusing. The evaluator made the mistake of forgetting to generate the metadata XML before submitting the metadata. Figure 3.5 illustrates how the flow of usage was unnecessarily confusing in version 2.

5.2.2 Compatibility Testing

Version 2 was only tested on the Mozilla Firefox browser during development. A compatibility evaluation was carried out by running the system using all the major browsers. The results are shown in Table 6.

Browser	Comments
Mozilla Firefox	Works as intended.
Internet Explorer	When adding item, the metadata editor only appeared in a small portion of the browser viewport. JavaScript in editors did not allow duplication of fields or proper generation of metadata XML.
Apple Safari	When adding item, the metadata editor only appeared in a small portion of the browser. XML preview of metadata does not work.
Opera	Nothing appeared besides the navigation menu. Pages had to be accessed individually. JavaScript in editors did not allow the duplication of fields or proper generation of metadata XML.

Table 6: Compatibility testing of version 2 under the most popular browsers.

This shows that there were severe problems with the compatibility of the system. This was found to be due to differences in the way browsers interpreted layout tags and also differences in the execution of JavaScript, especially with regards to DOM processing.

5.3 Evaluation of Version 3

To produce version 3, all the major problems with the system were fixed and a few new features were implemented. This section provides an overview of the second and most thorough iteration of evaluating the system. Even though most of the major problems were solved, the introduction of new features brought about a new set of problems, particularly with regards to usability.

5.3.1 Usability Testing

In order to evaluate the user interface of the system, user testing was conducted with ten volunteers. A heuristical evaluation of the user interface was carried out by following the guidelines provided by Nielsen and Molich (Nielsen & Molich, 1990). This involved having a small set of evaluators use the interface and judge its compliance with a recognised set of usability principles. Only ten users were chosen due to limited time and resources. However, according to Nielsen and Molich, a small set of evaluators is sufficient to find most of the

usability problems with a system and one does not gain significantly more information by using more evaluators.

All of the users had significantly high levels of proficiency with computers but most of the users were not familiar with the domain of the system. This meant that they could not be made to evaluate the system directly, on their own. Since the server management of CALJAX was not intended as a walk-up-and-use system, familiarity with the domain of the system was not considered to be crucial and would not affect the usability testing of the interface.

Each user was made to test the system separately. An overview of the system was provided to them, outlining the function of the whole CALJAX system. They were told that they would be testing only the central repository management interface. The purpose of the management system and the tasks usually performed in such a system were also explained. A small section also provided an introduction to the domain specific terms involved in the system. The users were asked to carry out a list of tasks which included operations on all the different sections of the system.

Unlike with traditional user testing, heuristic evaluation allows for a higher level of interaction with the evaluators. This meant that, at all times, the evaluators' actions were observed and if they had questions about the system these were answered. Answering domain specific questions helped users understand what they were supposed to do and helped them better assess the system with respect to the characteristics of the domain. After 15 minutes, the evaluators were asked to fill in a questionnaire that asked them to rate the system against the various heuristics. The evaluators also wrote down any specific problems they had encountered during the use of the system that justified the scores they had attributed to the various heuristics.

Heuristic	Average Score
Reasonable response time	9.1
Information in natural and logical order	6.8
Freedom to browse through system and cancel unwanted states	8.4
Consistent and followed conventions	7.0
Useful error messages	4.6 (5 evaluators)
Ability to cancel mistakes and prevention from committing errors	6.8
Minimal need to recall information	6.9
Clear which steps were available next	7.0
Flexible and efficient	8.0
Relevant and not unneeded information presented	7.3
Did not encounter anything unexpected	6.5

Table 7: Results of heuristic evaluation for version 3.

Table 7 shows the results of the evaluation. The lowest score for the answers was 1 and the highest score was 10. Higher scores indicated better usability for all the heuristics.

The table shows that the heuristic evaluation produced positive results for most categories. The first major problem that stands out is that those users who encountered error messages found

them to not be useful. While these errors were not critical system errors but simply users forgetting to fill in certain values, the messages produced confused the users more rather than helped them. This is discussed further when considering the comments the users provided.

The next lowest scoring heuristic was "Did not encounter anything unexpected". This had a strong correlation with those users who encountered error messages. These users found the messages confusing and therefore unexpected.

The low score on "Information in natural and logical order" emanated from one particular usability problem that was encountered by all the users of the system. This had to do with the presentation of the various schema editors under different metadata formats. Most users did not realise that each format had editors to choose from and clicked directly on the heading with that metadata format.

"Ability to cancel mistakes and prevention from committing errors" also gave a low average score which was possibly related to the fact that the system did not allow users to undo any actions taken. This could not be provided due to time restrictions.

"Minimal need to recall information" was a result of users having to remember the name of any items they had recently created or modified. When asked to edit a file they had recently created, several evaluators searched for the title specified in the metadata content rather than the name of the item they had uploaded.

More insight can be gained by considering the various comments about usability problems users encountered, as summarised below:

5.3.1.1 General

Error messages – The error messages presented were not clear to the evaluators. This was because they were non-specific (e.g., "Could not read metadata file: null." after a user submitted without choosing a valid file) and too technical (e.g., an error reported from the XML parser when a user submitted blank metadata). No unexpected system errors occurred during the testing. All the errors occurred as a result of the users forgetting to browse for a file to upload or forgetting to select an item to view/edit/delete.

5.3.1.2 Adding and Editing

Choices – This was the most common problem reported. The way the interface was laid out did not make it clear to all the users that they were presented with different kinds of editors from each format of metadata that was supported by the system. Each metadata format was listed followed by a list of editors. Some users clicked on the name of the metadata format while others did not know what to do. Even though there were instructions in the window, none of the evaluators noticed it, even when they ran into trouble.

Entering metadata – A few evaluators did not realise that they could increase the number of fields available. They entered two creator names separated by commas instead of clicking on a button provided to generate a new field. One user confused attributes with fields having the same name and thought that the editor (Schema based) was quite confusing. MODS records have a significant complexity and the editor does not enumerate or provide clues as to what each field can contain. Many attributes/less used fields take up a lot of space in the editor and users can get confused while editing more complex metadata.

Choosing file to upload or overwrite – Some evaluators could not figure out how to upload a file. They did not pay attention to the instructions about leaving the file upload field blank to keep the existing file in the repository unchanged. One user suggested that the position of the upload box did not provide an intuitive flow of presentation.

XML editing – None of the users opted for the plain XML editor.

5.3.1.3 Browsing/Viewing/Deleting

Search options – Most users found the search features to be useful while trying to locate file items. Few users browsed through the repository directly or entered item names. However, a few users attempted to enter metadata field information into the search control even though there were instructions warning that the search only searched item names. One user attempted to use wildcards which were not supported and resulted in search failures. Another user was confused about metadata content dates and the item modification date and suggested making the difference clearer.

Confirm delete – A few users did not confirm the deletion of a file they chose to delete.

Viewing – A user thought that plain display of XML for metadata was hard to understand and should be formatted properly before being displayed. Although some key fields are retrieved during the listing of items, there is no simple way of displaying the whole set of metadata without writing a custom tree viewer or a schema based tree viewer. Time constraints did not allow for this feature to be implemented.

5.3.2 Performance Testing

The components tested were unchanged in the final version. This is discussed in Section 5.4.3.

5.3.3 Compatibility Testing

All the functions were tested on these browsers: Internet Explorer (different versions), Mozilla Firefox, Opera and Apple Safari. This is shown in Table 8.

The problems with the execution of JavaScript in version 2 were fixed. Also, major layout problems were addressed.

Browser	Comments
Mozilla Firefox	Works as intended.
Internet Explorer	Some table borders are not as expected. XML preview of metadata does not work sometimes for unknown reason.
Apple Safari	XML preview of metadata does not work. Choice of editors table has no cellpadding.
Opera	Choice of editors table has no cellpadding.

Table 8: Compatibility testing under the most popular browsers.

5.4 Evaluation of Final Version

As many problems of version 3 were fixed as time allowed. The final version of the system was also evaluated. Time constraints did not allow for as rigorous evaluations as had been carried out for version 3 and, also, the final version featured only small changes over the previous iteration so these were not needed.

5.4.1 Usability Testing

Usability was carried out using the same procedure as described for the usability of version 3 of the system except this time only two evaluators tested the system. Also, the evaluators were given more time to test out the system than on the previous iteration so that they would be able to find smaller usability issues with the system since most of the most common issues were considered to have been fixed after the testing of version 3.

Heuristic	Average Score
Reasonable response time	9.0
Information in natural and logical order	8.5
Freedom to browse through system and cancel unwanted states	9.0
Consistent and followed conventions	9.0
Useful error messages	8.0
Ability to cancel mistakes and prevention from committing errors	7.5
Minimal need to recall information	9.0
Clear which steps were available next	9.0
Flexible and efficient	8.5
Relevant and not unneeded information presented	8.5
Did not encounter anything unexpected	9.5

Table 9: Results of heuristic evaluation for the final version.

Table 9 shows the results of the user testing of the final version. It can be seen that most of the scores have substantially increased. Principally, the low scores in version 3 were all corrected to produce positive scores.

Both evaluators found that the interface was easy to use and none of the evaluators required any help during the use of the system besides asking domain specific questions.

The system responded within reasonable time according to both evaluators. Since usability issues dealing with confusing layouts were fixed, the evaluators judged the system to present

information in a natural and logical order. As evaluators were encouraged to try out the different features of the system, they encountered different error messages. On the previous iteration, error messages were confusing and unclear and reported to be a major usability issue. One evaluator attempted to edit the XML content of an item but typed in invalid XML. The system did not update the item concerned and produced a helpful error message indicating that the metadata input was invalid. This justifies the good score for the system's ability to cancel mistakes and to prevent errors from being committed. The other evaluator typed in the name of an item wrongly and was shown an error message that indicated that such the item name specified was invalid. Both users found the error messages useful and this explains the greatly improved score for "Useful error messages".

The users did comment on some lacking features of the interface which were not noticed before. This included a comment about how indications of whether a task had been completed or failed was not clear enough. This was because after changing the contents of a file on the repository but leaving the metadata unchanged the evaluator had the impression that no changes had occurred as only the metadata was shown in the confirmation screen. The other evaluator suggested that it would be useful to be able to delete or edit items directly when browsing lists of items or viewing an item. Also one evaluator was not satisfied with the view of metadata provided as, at one point, for an unknown reason, this had to be refreshed manually to display the XML tree properly.

All the other scores show that the system was judged to have a good usability in general and this indicated that the goal of providing a management system with good usability was achieved.

5.4.2 Compatibility Testing

All the functions were tested on these major browsers: Internet Explorer (different versions), Mozilla Firefox, Opera and Apple Safari. This is shown in Table 10. Since version 3, the major changes to the schema based editors for both adding system were most liable to affect compatibility. These worked well on all the browsers. While each browser still had slight differences in layout, these were minor and could not be changed easily without affecting the layout on other browsers.

Browser	Comments
Mozilla Firefox	Works as intended.
Internet Explorer	Some table borders are not as expected. XML preview of metadata does not work sometimes for unknown reason.
Apple Safari	XML preview of metadata does not work. Choice of editors table has no cellpadding.
Opera	Choice of editors table has no cellpadding.

Table 10: Compatibility testing under the most popular browsers.

The only remaining problems are minor issues and all the JavaScript and layout problems from the initial versions were no longer present in the final version of the system.

5.4.3 Performance Testing

The only part of the interface that required performance testing was the search function. The experiment described below was carried out to measure the performance of this system.

Experiment Aim

To measure the performance of the search function implemented in the system.

Methodology

The search function goes recursively through all the files in the system looking for matches till it finds a sufficient number of matches to be displayed in a page. The means of evaluating this system was by carrying out several runs of the system using different collection sizes and measuring the time taken in the worst case search scenario. That is, the time taken to go through the whole repository if no matches were found was measured. This was measured by modifying the server code to note the system time when a server request was received and the time after all the results had been returned. The difference in times would give the worst case scenario search time for that collection. For each collection size, the experiment was repeated several times to obtain an average searching time. The server machine used featured a dual core 2.50 GHz processor and 2 GB of memory.

Results

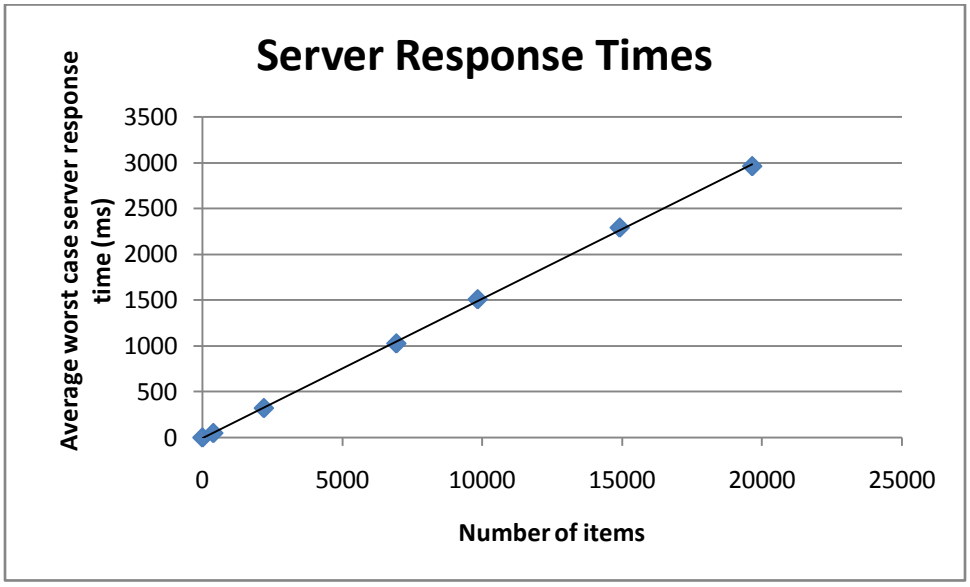

Figure 5.1: Server response times for different collections indicating the worst case scenario search time.

The average searching times for collections of different sizes are plotted on the graph shown in Figure 5.1.

Analysis

It can be seen that the search time increases linearly with the size of the collection. The maximum search time encountered was of the order of 3 seconds.

Discussion

The search function was tested on a collection that was larger than the Bleek and Lloyd Collection and produced a maximum search time of 3 seconds. This shows that the system is capable of dealing with real world, large collections. A search function using indices was beyond the scope of this part of the CALJAX system and the basic search function implemented demonstrates acceptable results.

Chapter 6

Conclusions

6.1 Discussions of Evaluations

For this project a digital repository management system was developed. The various critical and important features listed at the onset of the project were successfully implemented. The system was subjected to different types of evaluations and was shown to be an overall success.

6.1.1 Feature Support

As shown in Table 1, all of the 7 critical features that were necessary for a basic digital repository management system were implemented. All of the 3 important features were also implemented. While the features deemed to be of low relative important to this project were not implemented due to time constraints, those features listed as being desirable were mostly implemented.

6.1.1 Scalability

The system was tested on the Bleek and Lloyd Collection containing over 15000 items and was shown to be relatively scalable. The key factors of the system's scalability were the ability of its interface to break down large lists into various browsable pages and reasonable search times produced by the search function implemented.

6.1.2 Usability

Usability testing was conducted at the various iterations of the system development. The final usability evaluations showed that the resulting system scored high scores against all the heuristics used. There were a few minor issues with the final interface.

6.1.3 Performance

The only component of the system that required performance testing was the search function implemented. On large collections of around 20,000 items, the search times were below 3 seconds in the worse case. This was deemed to be a reasonable response time by evaluators.

6.1.4 Compatibility

The final version of the management Web interface was compatible with all the major Web browsers.

6.2 Future Work

6.2.1 Further Scalability Testing

Due to lack of time and resources some aspects of the system's scalability could not been tested. It still needs be investigated how such a server would respond to multiple simultaneous search requests.

6.2.2 More Sophisticated Search Function.

A more complex search function would be a significant improvement on the current system, based on inverted files which would be dynamically updated as documents are changed and would allow metadata content to be searched effectively.

6.2.3 Further Browsing Options

A browsing system that would allow sorting on metadata field contents would be a good enhancement to the current system.

6.2.4 List of Updates

Another potential problem regarding the performance of the server would be its response time when returning the list of items that have been updated to an offline repository since it was created. For this project, it was assumed that these would be of small size and since they consisted only of metadata would be manageable. A production system would require an AJAX based streaming of parts of this list.

6.2.5 Use of XML Schema

A useful improvement to the system would be making full use of all aspects of the XML Schema specification for the Schema based metadata editors. This includes the use of enumerations of possible values of certain fields, annotations found in the Schema description and input validation based on field restrictions also described in the Schema specification.

Bibliography

Apple. (n.d.). *Internet & Web: Web Content*. Retrieved October 2009, from Developer Connection: http://developer.apple.com/internet/webcontent

Coombs, K. A. (2007). Building a Library Web Site on the Pillars of Web 2.0. *Computers in Libraries , 27* (1).

Coyle, K. (2004, July). *Rights Expression Languages: A Report on for the Library of Congress.* Library of Congress.

Doernhoefer, M. (2006, July). Surfing the Net for Software Engineering Notes - JavaScript. *ACM SIGSOFT Software Engineering Notes , 31* (4), pp. 16-24.

EPrints. (2009). Retrieved October 2009, from EPrints: http://www.eprints.org/

Fedora Development Team. (2005). *Fedora Open Source Repository Software, White Paper.*

Flanagan, D. (2002). *JavaScript: the definitive guide* (4th Edition ed.). O'Reilly.

Garret, J. J. (2005, February 18). *Ajax: A New Approach to Web Applications*. Retrieved May 2009, from Adaptive Path: http://adaptivepath.com/ideas/essays/archives/000385.php

Greenstone Digital Library Software. (n.d.). *Factsheet*. Retrieved May 28, 2009, from Greenstone Digital Library Software Web Site: http://www.greenstone.org/factsheet

Hedstrom, M. (1998). Digital Preservation: A Time Bomb for Digital Libraries. *Computers and the Humanities* , 189-202.

Hillmann, D. (2007, November 7). *Using Dublin Core*. Retrieved May 28, 2009, from Dublin Core Metadata Initiative: http://dublincore.org/documents/2005/11/07/usageguide/

Körber, N., & Suleman, H. (2008). Usability of Digital Repository Software: A Study of DSpace Installation and Configuration. *11th International Conference on Asian Digital Libraries, ICADL 2008*, (pp. 31-40). Bali, Indonesia.

Lucy Lloyd Archive and Resource and Exhibition Centre. (2009). Retrieved from Lloyd Bleek Collection, University of Cape Town: http://www.lloydbleekcollection.uct.ac.za/index.jsp

Maslov, A., Mikeal, A., & Legett, J. (2009). Cooperation or Control? Web 2.0 and the Digital Library. *Journal of Digital Information , 10.*

Network Development and MARC Standards Office of the Library of Congress. (2009, September 25). Retrieved October 2009, from Metadata Object Description Schema: http://www.loc.gov/standards/mods/

New Zealand Digital Library Project, University of Waikato. (2009). *Greenstone Factsheet*. Retrieved May 2009, from Greenstone Digital Library Software: http://www.greenstone.org/factsheet

Nielsen, J., & Molich, R. (1990). Heuristic evaluation of user interfaces. *ACM CHI'90 Conf.*, (pp. 249-256.). Seattle, WA.

Open Archives Initiative. (2008, December 7). *The Open Archives Initiative Protocol for Metadata Harvesting*. Retrieved May 28, 2009, from Open Archives Initiative Web Site: http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm

O'Reilly, T. (2007). What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. *Communications and Strategies* (65), 17-36.

Organisation for Economic Co-operation and Development (OECD). (2007). *Participative Web and User-created Content - Web 2.0, Wikis and Social Networking*. OECD Publishing.

Staples, T., Wayland, R., & Payette, S. (2003). The Fedora Project, An Open-source Digital Object Repository System. *D-Lib Magazine , 9* (4).

Suleman, H. (2003). Metadata Editing by Schema. *7th European Conference on Research and Advanced Technology for Digital Libraries (ECDL2003)* (pp. 82-87). Trondheim, Norway: T. Koch and I. T. Solvberg (eds).

The DSpace Foundation. (2009). *DSpace Manual*. Retrieved May 2009, from DSpace Website: http://www.dspace.org/1_5_2Documentation/index.html

W3C. (2005, January 19). *Document Object Model (DOM)*. Retrieved October 2009, from W3C Architecture: http://www.w3.org/DOM

W3C. (1998). *Extensible Markup Language (XML) 1.0*. Retrieved October 2009, from W3C Recommendations: http://www.w3.org/TR/REC-xml

Appendix A

User Evaluation Survey

User Testing of CALJAX Repository Management Software

This research is conducted by Suraj Subrun (Department of Computer Science, University of Cape Town) as part of a Computer Science Honours research project.

Consent Form

The information collected in this questionnaire is strictly confidential. No personal information is gathered. Your answers will be used to produce a report on the evaluation of the CALJAX Repository Management system.

Please note that during this experiment, your performance with the system will **not** be measured.

☐ Your participation in this experiment is voluntary. You may withdraw from the experiment at any time if you wish to do so and no explanation on your part is required. You are welcome to ask any questions or raise any concerns you have about the experiment now or at a later stage.

☐ You are over 18 years of age.

Thank you for participating. Your feedback is greatly appreciated.

Overview of CALJAX Repository Management Software

Introduction

CALJAX is a digital repository system with the aim of providing distributable copies of information collections. It consists of a central repository from which the distributable copies are created. These can be run offline from different types of storage media (such as DVD-ROMs or flash drives). They can also be synchronized with the central repository to retrieve any content that has been updated since their creation.

Management Interface

You will be testing the management interface to the central repository. This is aimed at providing a way of creating new content, editing existing content and browsing through the repository. It also provides a front-end for the distributable copies to retrieve any content that has been updated since they were created.

Repository Structure

Repository items can be any type of digital file (e.g., images or documents). Each item has an associated metadata file which contains information about the item (e.g., its creation date, author, etc.). Metadata is stored in XML files. There are many standards for storing metadata (e.g., Dublin Core, MODS) and the repository can be set up to work with several of these.

Additional Information

You will have 15 minutes to carry out a few tasks. When you have completed these tasks, you will be allowed to use the system in the remaining time. Please feel free to ask questions. After the 15 minutes, you can fill in the provided questionnaire.

Tasks

1. Browse to page number 2, and view the first image on the page.

2. View the following items:

    (a) ubolt.jpg.

    (b) An item from the Wilhelm Bleek notebooks.

    (c) Any item that was modified within the last week.

3. Find the list of items that were updated since 18th October 2009.

4. Add the item "sks.bmp" (you will be shown where it is located) to the repository. Use Dublin Core metadata. The title of the image is "Sunset", the description is "Gathering energy at sunset". The image has two creators: "Suraj Subrun" and "N. Karimbocus".

5. View the item you have just created.

6. Change the description of the item to "Gathering energy at sunrise" and change its title to "sunset". Do not change the image you have uploaded already.

7. Delete the item you have created.

8. Edit the repository item "eula.txt". Change its title to "Moddy title 2". Change the file to the new file "to do.txt" (you will be shown where this is located also).

PLEASE CIRCLE THE MOST APPROPRIATE ANSWER.

1. The system responded within reasonable time.

1 2 3 4 5 6 7 8 9 10

Strongly Disagree Strongly Agree

2. The system made information appear in a natural and logical order.

1 2 3 4 5 6 7 8 9 10

Strongly Disagree Strongly Agree

3. You felt free to browse through the system functions and to cancel unwanted states of the system.

1 2 3 4 5 6 7 8 9 10

Strongly Disagree Strongly Agree

4. The terminology used in the system was consistent and followed expected conventions.

1 2 3 4 5 6 7 8 9 10

Strongly Disagree Strongly Agree

5. In case you encountered a system error, you were presented with a useful error message.

1 2 3 4 5 6 7 8 9 10

Strongly Disagree Strongly Agree

6. The system provided prevented you from committing errors and enabled you to cancel unintended mistakes.

1 2 3 4 5 6 7 8 9 10

Strongly Disagree Strongly Agree

7. While using the system, you did not have to recall information from previous pages.

1 2 3 4 5 6 7 8 9 10

Strongly Disagree Strongly Agree

8. The presentation of the system made it clear which steps were available to you next at all times.

1 2 3 4 5 6 7 8 9 10

Strongly Disagree Strongly Agree

9. The system was flexible and allowed to you carry out the tasks efficiently.

1 2 3 4 5 6 7 8 9 10

Strongly Disagree Strongly Agree

10. The system contained information that was relevant and did not contain unneeded information.

1 2 3 4 5 6 7 8 9 10

Strongly Disagree Strongly Agree

11. While you were using the system, you did not encounter anything unexpected.

1 2 3 4 5 6 7 8 9 10

Strongly Disagree Strongly Agree

12. Any additional comments or criticisms about your experience with the system. Please feel free to elaborate on any problems or lackings that you have noticed with the system.