Customisable Abstract Representation Layer for Digital Libraries

CSC500W

Fu-Yao Kevin Feng
ffeng@cs.uct.ac.za

Supervised by
Prof. Gary Marsden

June 1, 2006

Abstract

The user interface is a very important component in a piece of software as it is the layer which allows user interaction with the underlying functionality. Within the domain of digital libraries modification to the interface layer, to make it more appropriate for target users, requires substantial programming skill.

This research studies the possibility of making a user customisable interface system by using HCI methodologies for user requirements identification and evaluation, as well as AJAX (Asynchronous JavaScript and XML) for design and development. The final prototype allows users to directly design pages by adding, deleting, dragging and dropping elements in a Web browser. The research ends with an expert evaluation of such a system where satisfactory results were shown.

Contents

List of Figures

List of Tables

List of Illustrations

Chapter 1

Introduction

A Digital library is an electronic system of information storage where Web services are provided for information seeking needs. To date, most research in this area has focussed on the efficiency and functionality provided by digital libraries, but less can be found focusing on the user interface and the usability aspects of accessing that functionality. [13]

For digital libraries, we can split users into two groups: namely system administrator and system user. System administrators use the functionality provided by the system to set up an environment which is usable to those accessing the digital library. These system users are then the final users who engage with the software, but have no control over the functionality provided or how that functionality appears.

The user group targeted by this project is the system administrator (henceforth referred to as 'user' in this document), who will use the system to set up a user interface layer for digital libraries.

The user interface of software is the layer through which end users interact with the underlying system. If poorly designed, the underlying system can be unusable and functionality cannot be accessed. Modifying the interface layer of current digital library software packages requires knowledge of programming, or at least markup languages like HTML and CSS. As most digital library administrators are likely to be librarians or those who merely manage information, it is unlikely that they posses these skills. The result is that administrators are less able to customise the user interface and thus make it more usable to their target audience.

In this project, the *evolutionary prototyping* methodology from software engineering [28] and different evaluation methods from human-computer interaction (HCI) will be adopted during the development. The development process is divided into three cycles where each cycle has a main objective to achieve; a prototype is also produced at the end of each development cycle and is then evaluated before entering the next phase.

Finally, to ensure the separation between interface software and digital library software, we will adopt a component-based architecture. Componentisation is a current trend in software development; with this architecture, it is possible to replace components in a system seamlessly. This project, *Customisable Abstract Representation Layer for Digital Libraries* (CARL), belongs to a larger project, *Flexible Digital Libraries*(FDL), where each project under it is a separate component. Therefore, a digital library can be constructed by establishing a collection of these components. This project aims to develop a UI component for project FDL, but is also striving to design the system in a way so that connections to other digital libraries are possible.

1.1 Problem description

Of particular interest to this project is open-source digital library software as this affords us the opportunity to modify the interface, without having to implement our own digital library back end. The user interfaces of open source software projects, however, are in general perceived to be hard to use or difficult to customise [21]. For example, to change the look and feel of DSpace [8] (a popular open source digital library package), requires that users have knowledge of Java/JSP programming and also the structure of DSpace itself. Similarly, in Greenstone3 [2] (another popular open source digital library package), interfaces are dynamically generated by Extensible Stylesheet Language Transformation (XSLT), requiring knowledge of both XML and XSLT before customisation can be made. For digital library administrators who have no background in computer science, it will be very difficult to do customisation.

Most digital libraries are based on Web technologies, thus the rendering of interfaces depends on Web browsers. Not wishing to put a further burden on digital library administrators, we decided to develop our software to execute within a standard Web browser. This poses another problem: although W3C had announced various standards for Web technologies, different browser vendors chose to support different subsets of

some standards, thus different Web browsers render pages differently. One example is W3CDOM [34], it is a platform/interface which allows programs to dynamically access and update contents of a document, such as an HTML document; but different Web browsers support different subsets of it and functions supported to access the document are also different. For instance, to acquire a reference to an element in a document, method *getElementById(Object_ID)* is used in Firefox while method *all(Object_ID)* is used in Internet Explorer. To overcome this problem, some Web developers choose to develop different versions of software for different browsers and some choose to develop software that is compatible with most browsers.

For the development of this project, we decided to make the software executable in any browser. However, it soon became apparent that preserving compatibility was going to absorb a disproportionate amount of time. Thus it was decided to develop a Firefox compatible version of the system, because both this research and Firefox are open source projects.

Since the introduction of W3CDOM and the increasing level of maturity in JavaScript and CSS, it is possible now to create an interactive environment in a Web browser, instead of static HTML pages. Together with other new technologies, such as XML and XSLT, the idea of user customisable interfaces that allow interactive positioning of elements within Web browser becomes possible. Thus digital library administrators can easily modify the interfaces without touching source code.

This possibility is just part of a wider move in Web applications called AJAX, which breaks the synchronisation in request-response actions of conventional Web applications. All user actions which do not require server processes are taken care of by the Web browser itself. AJAX is essentially the combination of some Web technologies which enable the dynamic manipulation of contents of Web documents – these technologies include JavaScript, XML/XSLT, CSS and HTML.

1.2 Reason for developing customisable user interface

Software developers are often not HCI expert themselves; interfaces designed can sometimes be hard to use by the target group of software users. The same is true for digital

libraries. So a possible solution to overcome the problem is to allow users to define and design their own interfaces. Though allowing this freedom, the system also requires users to perform some configurations, such as page properties, Web services properties and site navigation properties, in order to best optimise the usability of the final interface.

Through the advancement of Web technologies, many features not possible in the past can now be performed in a Web browser. Though technologies are now present, little has been done in the field of digital libraries to apply them to the field of interface creation. Thus this project tries to make use of current Web technologies and implement an idea that can be useful in future to all other digital library packages.

The reasons for making this system to run in a Web browser are: platform independence; universal accessibility and the degree of familiarity of environment. The idea of making these interfaces in a Web page editor, such as Dreamweaver, by extending them with appropriate plug-ins , will sacrifice universal accessibility and platform independence. If the system is Web-based, the working environment for the system will be similar no matter what setup is used by client side machines.

1.3 Project objectives

The broad aim of this project is to survey different combinations of platforms, such as Web browsers, digital libraries and Web technology, to develop a Web based system which supports user interface customisation. There are two main objectives in this project:

- Customisable Interface
  The main objective of this project is to enable users to design interfaces freely within a Web browser without the need for writing code. A number of pages can be defined and designed to form a complete set of interfaces for a digital library. The interface will allow the user to reflect-out the underlying functionality of the digital library, and configure the visual appearance of elements such as the representation of search/browse results, number of results per page and paging style.

- Interface abstraction
  To achieve abstraction, a middle layer between the interface component and the back-end digital library is designed, where all communication between the two parties is processed. This objective only serves to show the feasibility of such architecture, since the request and response structure of different digital libraries works differently, a new middle layer may need to be designed for a different digital library, so the interface component can function properly. The configuration of Web services mentioned above may also need to be redesigned to correspond to Web services provided by a different digital library.

1.4 Plan of report

The plan for the remainder of the report is as follow:

- Chapter 2: Background and Related work
  This chapter can be roughly divided into three sections: digital libraries; HCI evaluation methods and Web technologies. The digital libraries section includes concepts of digital libraries and the basic structure of different digital library packages. The HCI evaluation section describes basic concepts of HCI and how to extract useful information from different evaluation methods. The Web technologies section discusses different technologies used in this project, such as JavaScript, XML and XSLT, in detail.

- Chapter 3: Development Cycles
  The design and implementation of this project is divided into different stages, where each stage has a different objective. These development cycles are discussed here in detail. The preparation made before design, implementation processes of the system carried out, decisions on what development environment to use and the splitting of different development stages are also discussed.

- Chapter 4, 5, 6: Phase 1, 2, 3
  These three chapters include details of each implementation cycle. Each chapter focuses on the objective of the cycle, implementation details, evaluation of prototypes, and the effect of evaluation results on the design of the next development cycle.

- Chapter 7: Discussion and Conclusion
  This chapter concludes the findings of this project and discusses the future work if this work is to be extended.

Chapter 2

Background and Related work

2.1 Digital Libraries

Digital libraries are complex information systems, whose purpose is to satisfy user needs such as information seeking, organising and managing, and communicating with users and their agents. Another aim of digital libraries is to preserve the integrity of information and to ensure the persistence, over time, of collections of digital works.

In simple terms, a digital library stores and preserves information, and at the same time, provides services which allow users to collect their desired information. The type of information that can be stored in the digital library includes documents, images, videos, audio, 3D models, etc. Services, such as search and browse, provide ways to retrieve information from a digital library.

The layer of *User Interface* is essential in software engineering, where it is the layer which comes in touch with users directly. It must be clear and intuitive, allowing users to complete tasks with minimal problems. At the moment, the layer of user interface in digital libraries has not received enough attention; since the majority of digital libraries are open-source projects, this nature of open-source software often leads to low usability [21]. Thus digital library software, such as the Greenstone Digital Library [2] and DSpace [8], have not had their usability studied and developed to a level that may have happened had they been commercially developed.

It is interesting to note that there has been little research into providing interfaces for those creating or configuring digital libraries. There has certainly been a lot of

work on interfaces to support those searching or browsing a collection, but little work in helping librarians to create the collection in the first place. The main exception is the work of Bainbridge et al. [1] who outline a system which allows librarians to collate information into a single collection. This system, however, has limited impact on how the collection will appear to the final user. Another system was the component assembly system of project *Flexible Digital Libraries* by L. Eyambe [7]. This system uses a component assembly approach to install and configure digital libraries in a very high level fashion using Graphical User Interface (GUI). One similar system we could also find was one which permitted customisation for deployment on mobile, small-screen, terminals [17]. Rather than creating a flexible, general purpose tool, this system acted as a wrapper around the Greenstone 2 configuration macros and was thus much limited in functionality. Furthermore, the system was developed as a stand-alone MS Windows application.

In the past, digital library systems were very often monolithic [29], and custom-built from scratch. No code re-use was enforced and modification/customisation to the system was difficult. Interoperability between systems was also difficult to achieve. The maintainability and extensibility of the system were also decreased as the system complexity increases [7].

It is now widely accepted as good software engineering practice that software be developed in component form [39]. Thus, in 1994, the discussion on the future of digital libraries [11] concluded that components are an integral part of the solution into the development of digital libraries [29]. In the early age (90's), although digital library software was developed in component form, they still lacked interoperability amongst components because there was no single framework for the development of the component, meaning that components weren't able to communicate with one another.

2.1.1 Open Digital Library

Open Archive Initiative (OAI) was established in 1999 to address the issue of interoperability. The Protocol for Metadata Harvesting (OAI-PMH) [4] was introduced by OAI as the *glue* to attach the various bits and pieces together. Thus, by adopting OAI-PMH protocol, components, though developed by different developers, would be able to communicate with one another. This framework provided by the OAI was then

used to develop a series of components interconnected with each other. This network of components is thus called an Open Digital Library (ODL) [31]. ODL components are available freely over the Internet and it is possible for one to build a digital library by putting a few of these components together.

Although ODL has no user interfaces, but a set of machine interfaces was defined, which allows these components to communicate to each other. However, a GUI component was written for ODL to provide basic access to these Web services components – this GUI component made use of the consistent machine interfaces to design separate sets of interfaces for each component and make connections in between. When this component was developed, developers were not Human Computer Interaction (HCI) expert and did not focus on it; the distributed nature of ODL also adds a level of complexity to the development of the user interface. Thus there exists inconsistency in these interfaces provided by this GUI component and the usability level of this GUI component is not up to standard. This still holds true and thus HCI principles need to be applied when designing these interfaces.

2.1.2 Greenstone Digital Library

The Greenstone Digital Library (GSDL) is an Open-Source digital library software system developed by the New Zealand Digital Library Group in the Department of Computer Science at the University of Waikato, New Zealand. The reason behind the development of the Greenstone project was that the designers wanted to make on-line technical reports more accessible to the research community by presenting them over the Web in a uniform, and fully-searchable way [37]. The aim of Greenstone is to build, maintain and serve digital collections; these collections may contain documents (text, html, pdf, etc), images, audio and video. It was developed for a wide variety of platforms: Windows, Macintosh and Linux. The features provided by GSDL include full-text mirroring, indexing, searching, browsing and metadata extraction [21].

There are two categories of user in GSDL: those who build collections, i.e. administrators, and those who access collections, the library users [21]. GSDL provides three different ways of building collections: Command-line, Collector (Web-based) and Greenstone Librarian Interface (GLI)(see Figure 2.1). In the user manual of GSDL, it suggested that the librarian interface should be the preferred interface for building

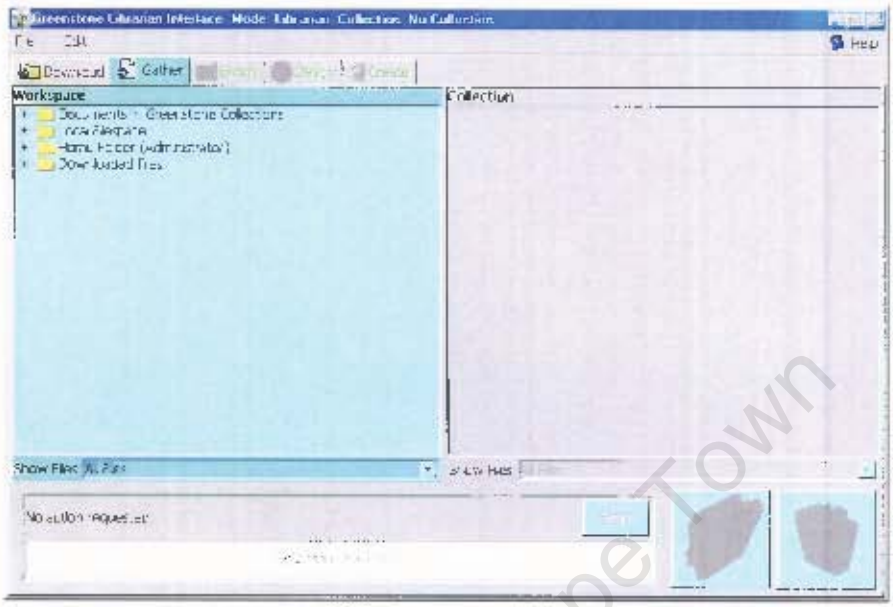

Figure 2.1: Greenstone Librarian Interface

collections [6]. Whilst Greenstone may not suffer the interface problems of component-based systems (inconsistent interfaces), much work is still to be done on the Greenstone interface to realise its full potential.

In the paper *Usability and open-source software development* [21], 23 fourth and fifth year Computer Science students were tested on building collections in GSDL using both Collector and Command-line instructions. The participants were given both a paper-copy and an electronic copy of the manual. The researchers concluded that typical usability issues (the difference between users and developers) were discovered and although it benefited from the contribution of many people (open-source software) these usability issues had never been addressed. They suggested that open source software (digital libraries in particular) may need to put more effort into interface design if they are to produce software for the desktop of the typical user.

2.1.3 Flexible Digital Libraries

Flexible Digital Libraries (FDL) is a digital libraries project hosted in Department of Computer Science at the University of Cape Town in South Africa. The aim of the project is to investigate techniques, models and tools for constructing digital libraries

in a more efficient and effective way.

Based on simple components connected in a network of services, the project is divided into several areas of research:

- A visual component that composes modules into complete systems and supports the specification of communication between components
- Packaging of the system for fast and easy deployment
- User interface components for easy and intuitive building of interfaces for digital libraries
- Supporting scalability of digital libraries

This project (Customisable Abstract Representation Layer for Digital Libraries) is the *User interface component* part of the Flexible Digital Libraries project, but at the same time this project is also related to Human Computer Interaction (HCI), since HCI techniques are required to design and implement a *usable* interface.

2.2 HCI Methodology

The aim of Human Computer Interaction (HCI) techniques are to improve the interaction between user and the user interface of the software they are using. HCI is a very abstract and broad field, which includes many fields of study, such as psychology and anthropology. In this project, we will only be concentrating on the usability, design and evaluation parts of HCI.

When designing the user interface for a piece of software, the most important consideration is how the user can interact with the software; yet this is often neglected by software developers. HCI provides techniques for identifying the needs of users more accurately than, say, simple interviews. The one we have chosen to follow is called *participatory design* [18]. Participatory design is based on workshops where developers, users and business representatives work together to design a solution [5]. Having users participate in the design process ensures a usable design, which also prevents the need to alter a design at a later stage of development.

There are other user-centered design methods in HCI which help to identify user needs, such as *task analysis* and *contextual inquiry*. Task analysis involves gathering of user tasks and break them down into their constituent subtasks and operations; analyses are then carried out to construct a basic structure with sequential activities [36]. Contextual inquiry is a process whereby users from the target user group of the system are interviewed at his/her workplace(s). The user being interviewed is considered as an expert in the work situation. The interviewer takes note of what the user does and says in order to construct a design of the system [36].

Participatory design method was chosen over the other methods for the reason that we want users to be directly involved in the design process of the system, thus to ensure a usable system design for the target user group. In this way, we also minimise the time required for the design process in order to produce a feasible design.

There are many ways to evaluate the user interface depending on circumstances. These include techniques such as Heuristic evaluation, Observational evaluation and Ethnographic evaluation.

2.2.1 Ethnographic evaluation

Successful re-designs depend on an accurate and thorough evaluation of the existing system [23]. Techniques such as observation, hands-on experience and questionnaires are often used to evaluate a system; but in many cases, the developers lack the knowledge of who will be using the system and how the system will be used; they simply develop the software in the way they think it should be done.

In order to understand how users interact with the system in real life, it is essential to study the social context of the work of the user. This requires the interface designer to observe the user in *real life* – seeing how a user uses the system. From this, more real-life situations can be studied than in a *control room* method where users are called to perform some predefined tasks. This method of observing users is called *ethnographic evaluation*.

In [14], four ethnographic evaluation methods are stated:

- Concurrent ethnography: where design is influence by an on-going ethnographic study taking place at the same time as systems development.

- Quick and dirty ethnography: where brief ethnographic studies are undertaken to provide a general but informed sense of the setting for designers.
- Evaluative ethnography: where an ethnographic study is undertaken to verify or validate a set of already formulated design decisions.
- Re-examination of previous studies: where previous studies are re-examined to inform initial design thinking.

The use of these methods should not be seen as mutually exclusive – some of the uses could be harnessed together.

In [23], a few guidelines were derived for preparing for the evaluation, performing the field study, analyzing the data and reporting the finding:

1. Preparation
    - Understand organisation policies and work culture.
    - Familiarise yourself with the system and its history.
    - Set initial goals and prepare questions.
    - Gain access and permission to observe/interview.
2. Field Study
    - Establish rapport with managers and users.
    - Observe/interview users in their workplace and collect subjective/objective quantitative/qualitative data.
    - Follow any leads that emerge from the visits.
    - Record your visits.
3. Analysis
    - Compile the collected data in numerical, textual and multimedia databases.
    - Quantify data and compile statistics.
    - Reduce and interpret the data.
    - Refine the goals and the process used.

4. Reporting

   - Consider multiple audiences and goals.
   - Prepare a report and present the findings.

It is the virtue of ethnography that it seeks to present the portrait of life as seen and understood by someone who lives and works in the domain concerned [14]. The intention is to observe the participants through the day-to-day activities they accomplish within their socially organised domain. Thus we gain in-depth knowledge of how participants interact within their social domain.

Ethnography also has its problems: the data collected from the evaluations are often qualitative and thus it is difficult for the designer to understand the data and make a change to the existing system. Therefore, many software engineers feel that ethnography is far too unsystematic; the results presented are abstract, design options are not clearly stated and it does not attend sufficiently to engineering needs [14]. From this aspect, the power of ethnography becomes insignificant.

In this research, ethnography is particularly useful in observing how evaluation participants interact with the system prototype. Through observations, it is possible to seek out design flaws and put the focus on important sections that need redesign.

Since prototyping is used in this research, ethnography also becomes an advantage. It reduces refinement cycles and helps to identify questions and problems for the next phase of system study, which can be used by the system designer [28].

2.3 Interface design

The idea of a Graphical User Interface (GUI) was introduced as early as 1963 when Ivan Sutherland demonstrated *Sketchpad* [32] in the course of his PhD thesis. Underlying the power and usability of graphical user interfaces is the notion of direct manipulation, defined by Ben Shneiderman, which involves continuous representation of objects of interest, and rapid, reversible, incremental actions and feedback [27]. Drag and drop is one of the techniques used to achieve direct manipulation. It is the action of clicking on a virtual object with a pointing device, such as a mouse, and dragging it to a different location, or onto another virtual object. An example of using drag and drop in a

windows system would be the re-positioning of a window: a user can click and drag a window to a desired position before releasing the mouse button. In the intermediate positions of a dragging action, the contents of the window being dragged is constantly redrawn.

Thus drag and drop is an important technique in direct manipulation of interfaces, which allows users to achieve tasks by using their pointing device, but without the need for issuing explicit commands. Though it was initially popularised in Mac-OS, this mechanism is now a common technique used when developing software. Operating systems such as Microsoft Windows, Linux X-Windows and Apple OS all support drag and drop, thus software developed for these platforms are likely to inherit this functionality as well.

In this research, drag and drop is also implemented to allow users to drag elements around in a Web browser. We believe this technique to be more intuitive than command line interfaces and it should be familiar to our target user group. Although complex, the dragging and dropping functionality is achieved by using some third party JavaScript libraries; more details will be explained in latter sections of this chapter.

2.4 AJAX

Asynchronous JavaScript and XML, or AJAX, is not a technology itself, but a combination of several Web technologies to create interactive Web applications [10]. These Web technologies include:

- XHTML (HTML) and CSS for standard data presentation and page styles
- Document Object Model (DOM) for dynamic content interaction/manipulation
- XMLHTTPRequest object to asynchronously exchange data with Web server
- XML and XSLT for data interchange and manipulation
- JavaScript for linking everything together

2.4.1 AJAX architecture

In the classic model of a Web application, a user triggers an action by making an HTTP request to a server – the server then prepares results by executing some processes, such as retrieving data from database, number calculation or request to some other legacy system. Once available, the result (in the form of an HTML page) is returned to the user. This model is an adapted version of original hypertext model; while it is good for hypertext, is not so good for Web applications.

From a technical point of view, this model makes perfect sense since the Web server takes care of each request from the user. Unfortunately, this model does not necessarily make for a great user experience, because each request-response cycle takes some time (while the server performs its calculations, the user can't do anything but wait).

In an ideal world, once an interface is loaded completely, the user should not be held back in the process of performing tasks; in fact, the user should perceive the application running on the local machine and not see application go to the server at all [10]. The repetitive nature of start-stop-start mechanism must be eliminated to create a better user experience, and this is where AJAX comes into play. By introducing AJAX as the intermediary layer between the server and the user, this layer is responsible for rendering interfaces and making requests to the server on behalf of the user. In [10], Figure 1 shows the two different models for comparison.

The start-stop-start mechanism is therefore broken into an asynchronous nature. Normal HTTP requests from users are replaced by JavaScript calls to an AJAX engine. All responses to user requests, where a trip back to the server is not required, are processed by the AJAX engine. These requests include data validation, interface changes or even navigation. If data is needed from the server, the engine makes the request asynchronously in the form of XML, without stalling the user's interaction with the application. In this way, some of the workload is shifted to the front-end, so waiting time is minimised and the user is given an unbroken experience of interaction. In [10], Figure 2 shows the two different models in how requests are processed.

2.4.2 Real life application

In the past two years, Google invested a great effort in developing AJAX applications. The latest products from Google, such as GMail, Google Suggests, Google Groups and Google Maps [12], are all AJAX applications. This demonstrates that the granularity of an AJAX application can be as small as a single function application, up to extremely complex applications, such as Google Maps [12].

Although there are advantages in using AJAX in a Web application, there are shortcomings too. These include:

- Usability problem: Can break behaviour of *back* button in a Web browser
- JavaScript must be enabled in Web browser
- Different Web browsers may not behave the same way
- Difficult to debug

Despite these limitations, we believe that AJAX technology provides the best platform on which to build a digital library interface editing tool inside a Web browser. Each of the components of AJAX are discussed in detail below.

2.5 XML and XSLT

Extensible Markup Language (XML) is a markup language designed to be capable of describing different kinds of data and facilitates the sharing of data across different systems (platforms), especially those over the Internet [25]. It is a subset of *Standard Generalised Markup Language* (SGML) with many SGML features. XML was designed and developed during 1996 and 1997 – XML 1.0 then became a W3C Recommendation on 10 Feb 1998.

2.5.1 XML structure

The basic structure of XML is a document similar to HTML, but much more flexible. Its tag structure allows it to do presentation, exchange or storage of data. XML takes

the form of a tree structure where each node can have a property list of its own. Illustration 2.1 is a simple list of class test results expressed using XML:

```
<?xml version="1.0" encoding="UTF-8"?>

<Test course="CSC115F" number="1">
    <title>Class Test 1</title>

    <question number="1" mark="15">Using Array</question>
    <question number="2" mark="10">Loops</question>
    <question number="3" mark="20">Inheritance</question>

    <result student="50">
        <passed>48</passed>
        <average>62</average>
        <highest>43</highest>
        <lowest>12</lowest>
    </result>
</Test>
```


Illustration 2.1: XML structure example

From this short piece of XML, we can see how flexible XML is in modeling data. The first line is the XML declaration stating what version of XML is being used and sets the encoding type. Then the rest consists of nested elements (nodes), some of which have properties (attributes) and contents. An element must have at least two tags, a start tag, e.g. <Test>, and an end tag, e.g. </Test>; in between the two tags, there can be any amount of text and elements. Thus XML takes the form of a tree structure.

It is mentioned above that XML is designed for data transfer. The following are some features of XML that makes it well-suited [25]:

- Format is readable to both human and machine
- Support for Unicode
- Represents data in the most general data structures: records, lists and trees

- Self-documenting format that describes structure and field names as well as specific values
- Parsing requirements and syntax are strict, which allows parsing algorithm to be simple, efficient and consistent

In some applications, XML can also have its weaknesses [25]:

- The verbosity of the syntax can be partially redundant. It lowers the human readability and application efficiency. The storage cost is also higher. In cases where bandwidth is limited, e.g. cell phones and PDA applications, the use of XML may be difficult.
- Since XML is essentially a tree structure, data is presented in a nested way. Thus a parser must be designed to recursively process nested data and also perform checks to detect improperly formatted data. This causes significant overhead in its efficiency; in the case of a resource limited environment, e.g. cell phone, PDA and embedded system, if XML input is fed from an untrustworthy source, resource exhaustion or stack overflow are possible.
- XML remained compatible to its ancestor, SGML, to some degree. Some would contend that this has resulted in a syntax which may contain a number of obscure and unnecessary features.
- Not many basic data types are supported in the parsing environment. This causes extra work before data can be processed from a document. For example, "1.23" cannot be specified to be recognised as a floating point number rather than a piece of text in XML. XML schema languages add this functionality.
- Modeling a non-hierarchical data structure requires extra effort.
- Mapping XML to relational model or OO paradigms is quite cumbersome.
- If XML is to be used as data storage, some argue that the file volume must be small.

2.5.2 XSL transformation

XSL Transformation (XSLT), or *Extensible Stylesheet Language Transformation*, is an XML-based language used for the transformation and rendering of XML documents. When an XML document is transformed, the original document is not changed, but a new document is generated based on the contents of the original document [16]. In this way, we can transform documents from one format to another.

The reason behind doing transformation is that when XML is used as the medium for data exchange (across the network), one would want to include only the raw data in the XML to minimise transferring time and data volume; but this data cannot be used as is when it is received, thus by applying transformation, XML can be transformed back to its original form, or to other usable format, e.g. HTML. In this way, data can become very flexible in terms of its format by using XSLT.

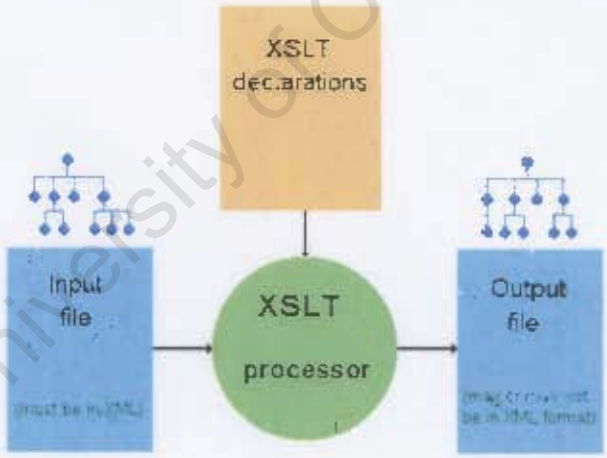

Figure 2.2: Transformation process

Figure 2.2 [35] demonstrates how XML documents are transformed. An XSLT processor takes a document (in XML) together with some XSLT declarations (in XSL), and generates a new output, which may or may not be in XML format. XSLT processor is usually an external program that carries out the transformation process.

XSLT is a declarative language, rather than an imperative language. XSLT stylesheets consist of a collection of templates, each of which specifies what to add to the result tree.

When an XSLT processor scans the source document (according to a fixed algorithm) and finds a node that meets the condition of a certain template, the transformation will take place [16]. Instructions within a template are performed in sequential order, but these instructions, ultimately, can include functional expressions, which in fact evaluate to some nodes to be added to the result tree.

2.6 Document Object Model

The Document Object Model (DOM) is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update contents, structure and styles of XML and HTML documents [34]. In this model, a well structured document can be represented in a tree structure, and in an object-oriented fashion. Thus, DOM itself is not so much a data structure, but a data manipulation tool that provides a simpler way of handling data.

Before DOM was standardised by the World Wide Web Consortium (W3C), many variants of it were implemented by different browsers. Since its main purpose is to manipulate data in a HTML document, different implementations may lead to different rendering results; this prompted the W3C to come up with a series of standard specifications for DOM; it is hence called *W3CDOM*.

In the year 2000, where Internet Explorer 4.x and Netscape 4.x were still widely used, the support for DOM standard from both vendors was still poor [38], thus interoperability was a great problem for Web application developers. Upon the launch of new versions of browsers from various vendors, the support for W3CDOM from these new browsers became significantly better. The two most commonly used Web browsers at the moment (2005) are Microsoft IE 6.x and Gecko based browsers (Mozilla and Firefox). Although both support a reasonable portion of W3CDOM, interoperability problem still exist due to different implementations of DOM within a browser. This problem remains a great one for developers.

2.6.1 DOM tree

When a document is loaded to a DOM tree, the tree resides as a whole in the computer's memory. The tree can be traversed through its nodes. Each node is an object

containing properties; the relationships between nodes may be parent, sibling or child. Thus, it is possible to navigate within the tree and make changes when necessary. The DOM of an HTML document can be accessed by means of Java/VB scripts, and updates to contents can be done by calling DOM methods using scripts. Together with Cascading Style Sheets and Java/VB scripts, this combination is called Dynamic HTML (DHTML) by some vendors [22].

Figure 2.3 illustrates the tree structure of the HTML code shown in Illustration 2.2 [26]. Whitespace nodes are excluded for simplification purposes.

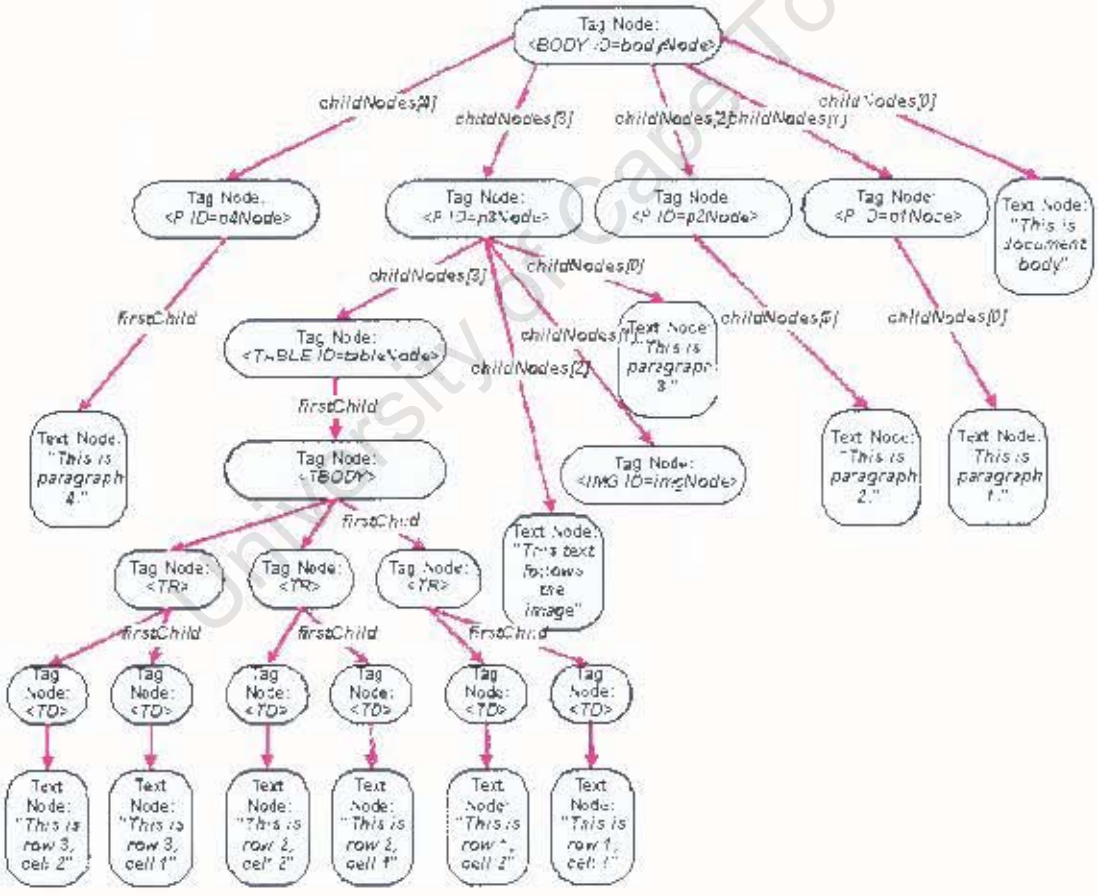

Figure 2.3: Tree structure of a complex HTML document

In Figure 2.3, **body** is the root of the document containing four **p** elements. The First, second and fourth p element contain text, while the third p element contains a more complex structure, consisting of text, image and a table. The relationships between

```
<html>
    <head>
        <title> DOM Demo </title>
    </head>

    <body id="bodyNode">
        This is document body
        <p id="p1Node">
            This is paragraph 1.
        </p>
        <p id="p2Node">
            This is paragraph 2.
        </p>
        <p id="p3Node">
            This is paragraph 3.
            <img id="imgNode" src="doc.gif">
            This text follows the image
            <table id="tableNode">
                <tr>
                    <td bgcolor=yellow>This is row 1, cell 1</td>
                    <td bgcolor=orange>This is row 1, cell 2</td>
                </tr>
                <tr>
                    <td bgcolor=red>This is row 2, cell 1</td>
                    <td bgcolor=magenta>This is row 2, cell 2</td>
                </tr>
                <tr>
                    <td bgcolor=lightgreen>This is row 3, cell 1</td>
                    <td bgcolor=beige>This is row 3, cell 2</td>
                </tr>
            </table>
        </p>
        <p id="p4Node">
            This is paragraph 4.
        </p>
    </body>
</html>
```


Illustration 2.2: A piece of HTML code

nodes are also presented in the figure.

2.7 JavaScript technology

JavaScript is an object-based scripting language well-known in Web development. It was originally developed by Brendan Eich of Netscape Communications Corporation under the name *Mocha*, then *LiveScript* and finally *JavaScript* [24]. Though given the name JavaScript, it has no real relation to Java. The similarities of both language are mostly in their syntax, because they are all derived from C. When compared to its counterpart, VBScript (only available in Microsoft Internet Explorer), the support for JavaScript from most browsers have made it the de facto choice for Web application developers.

JavaScript has no input or output; it purely relies on the host program in which it is embedded. Its best known usage is in Web technologies; other application programs such as Adobe Acrobat, Mozilla platforms and Dashboard Widgets in Apple's Mac OS X v10.4 have also embedded JavaScript interpreters to allow support for JavaScript [20] [3]. DOM is the interface through which JavaScript connects to applications (e.g. HTML documents); with this layer of connection, JavaScript can perform dynamic interactions with the DOM of the document, which was not possible in static HTML. Some examples of interactions are: checking input values; trigger functions on mouse events; mathematical evaluations and now even *Drag and Drop* effects. More on this will be addressed later in the section.

In this project, JavaScript and DOM are two key parts that have made dynamic content manipulation possible. Two Open Source JavaScript libraries, written by *Walter Zorn* and *Tim Taylor Consulting* respectively, are used in this project as enhancements to the Drag and Drop effects.

2.7.1 Walterzorn Drag and Drop library

Walterzorn Drag and Drop library [40] is a cross-browser JavaScript DHTML library written by Walter Zorn from Munich. This library enables drag and drop functionality to layers and images inside a browser; other functionality like resize, cloning and some simple styling of elements is also provided.

This library also allows developers a very flexible way of extending the core library to do some customised tasks. Four empty methods are provided inside the library, each of which is triggered under different events. These methods are my_PickFunc(), my_DragFunc(), my_ResizeFunc() and my_DropFunc(), where each name is self-explanatory. When an item is picked, dragged, resized or dropped, the corresponding function will be triggered and the developer can overload these methods to accomplish customised tasks.

2.7.2 Drag and Drop sortable list

This library is created by Tim Taylor Consulting [33], for the purpose of manually sorting an HTML list by dragging and dropping elements. This is particularly useful in this project when designing a navigation system or display of search results where the order of display of elements is important.

When an element is dragged inside a list, the element becomes semi-transparent and will exchange position with the element above or below depending on where it is dragged. Figure 2.4 shows how it is done.

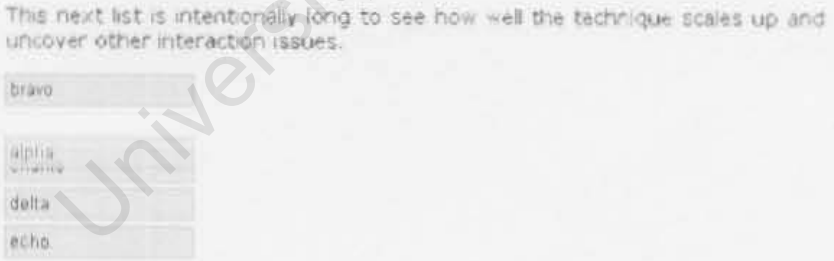

Figure 2.4: Sort list by dragging and dropping element

2.8 Summary

Three fields of study were discussed in this chapter, namely Digital libraries, HCI and Web technologies. These concepts serve as the background to different parts or processes of this research.

Digital libraries are fundamental to this research where they will serve as the back-end, providing *Web services* to the system. HCI concepts serve as a methodology for the design and evaluation processes of this research. Lastly, Web technologies are the tools used to implement the design. The focus of this research is on the combination of Digital libraries and HCI; with the aid of HCI methodologies, it is possible to design a visual component for Digital libraries that is useful to users.

Chapter 3

Development Cycles

It is always a difficult process for software users to express their real needs and requirements; it is also almost impossible to predict how users want to interact with the system and how the system affects their working practices. Although careful analysis and systematic reviews of requirements help to reduce uncertainty, it is always the best approach to allow users to try out some prototype before agreeing to a design.

Prototyping is one approach from software engineering to tackle this difficulty. A prototype is an early version of a completed system that can be given to users to demonstrate concepts and try out design options and to identify problems and possible solutions. In this project, the *evolutionary prototyping* approach is adopted. It is a form of Rapid Application Development (RAD) and Joint Application Development (JAD), where cycles of development are scheduled. A development cycle consists of different stages, such as objective specification, prototype function definition, system design, implementation and user evaluation. The user evaluation process evaluates the prototype produced from a development cycle. These results are then analysed and used as guidelines for design in the next development cycle. Thus prototyping is said to be evolutionary. After a few development cycles, the system will hopefully come very close to meeting the users' requirements.

Prototyping is followed because usability is one of the main focuses of this research. Having users involved in the development process, called *User Centered Design* (UCD) or *participatory design* in HCI, can dramatically improve the usability of the system. In each of the development cycles of this research, users were invited to participate in

the design and evaluation process.

Three development cycles will be carried out in this research, where different objectives are designed, implemented and evaluated in each cycle. The following sections will discuss these processes in detail.

3.1 Phase 0

This is the initial stage of the project, where research into possible solutions to the problem were carried out. The environment for the development of the system was designed; this involved the choice of digital library (as the back-end server), scripting language, development platform and server/client environment for the system. Different development phases were also identified, so each phase has a clear focus on what to achieve.

3.1.1 Problems

The following problems were identified:

- Which Digital Library software to use?
- What is the Server-side environment?
- What is the Client-side environment?
- Identification of development cycles

3.1.2 Methodology

Digital library

In order to choose a suitable back-end component for this project, a few digital libraries were considered. Two in particular were selected and tested. These were the Greenstone Digital Library (GSDL) and the Open Digital Library (ODL).

GSDL2 was initially looked at, for the reason that recent research on HCI in the Department of Computer Science at UCT used GSDL2 as the back-end server. Later

the attention was switched to GSDL3 after its beta release (in early 2004); the new architecture of GSDL3 was a more robust design where a component-based structure was adopted.

Components are called agents in GSDL3, where each agent is responsible for a different purpose. In [6], Figure 9 illustrates the architecture of GSDL3 for the rendering of interfaces and the handling of requests and responses. From the figure, MessageRouter is the agent which takes care of requests and responses, hence the name MessageRouter. All requests from the user are passed to MessageRouter, which then forwards these requests to the appropriate agent for processing. In the figure is also the Receptionist, which receives requests in the form of XML from the library servlet and then, according to the type of request, uses different action classes to forward requests to the MessageRouter in order to fulfill the process. When results (in the form of XML) are returned, sets of XSLT are applied to transform the results into an HTML page for presentation to the user.

The idea of using this structure as the back-end digital library is to design a component that sits on top, or works in cooperation with, these agents. After a lengthy development period, it was found that components in GSDL3 are tightly bonded to each other and it is difficult to design a component to work in-between the existing agents. It was decided that the amount of time that is required to get deep into the GSDL3 structure and design a component to work with its structure would be too much. Another possibility that may cause problems in later stages of development is that GSDL3 was still in its beta release at the time of coding – there may be unforseen problems with the software itself and the specification may change. Thus, at the time of writing, GSDL3 may not be a suitable digital library for the purpose of this research and the effort to build an interface system on top of GSDL3 was considered prohibitive.

The next package considered was ODL, which is similar to GSDL3 in its component-based design, but, however, is much simpler and the links in-between components are not as tight as in GSDL3. The developer of these components, Dr. Hussein Suleman, is also the head of the research lab in which this research was conducted, so support for ODL is at hand if problems are encountered. Thus writing an interface system (component) on top of ODL was deemed a sound choice.

Server-side environment

The server-side environment is designed to host the system after the digital library was decided. Since ODL uses an HTTP request/response model for its communication between components, an HTTP server must be set up and a scripting language chosen for the development of the system.

The scripting language must be decided first before choosing an HTTP server, since different HTTP servers support different scripting languages. Two mainstream scripting languages were considered, Java and PHP. Although Java is extremely well supported, PHP5 was chosen. The main reason for choosing PHP5 was its lightweight and as well as the good support for XML. XML is a crucial element in the design of this system, where responses from ODL components are all in the form of XML and transformation must be carried out. In PHP4, the support for XML was not particularly good and is not W3CDOM compliant; in PHP5, the support for XML was largely improved by support for the W3CDOM standard. Although Java has excellent support for XML, but due to its heavier weight, PHP5 is chosen for its lightweight and good support for XML.

The Apache HTTP server is one of the most popular open source HTTP servers, which is well known for its flexibility and ease of its installation and setup. This HTTP server is particularly popular when used with PHP as Apache supports it well. The Apache HTTP server is also very lightweight, which is also an advantage when considering system packaging and deployment.

Client-side environment

Since the system is a Web based application, Web technologies like HTML, Cascading Style Sheet (CSS), JavaScript and DOM are used. The aim of the project is essentially creating an interactive environment by manipulating Web contents in a browser, thus the use of JavaScript and CSS is inevitable.

The choice of browser for this project was not an obvious one. Initially, the development of this project tried to uphold cross-browser compatibility, at least for Internet Explorer 6 and Mozilla Firefox. As the project got further down in development stages, it was harder to keep the compatibility and much time was unnecessarily wasted. As Internet

Explorer's support for W3CDOM is not as good as that of Mozilla Firefox, FireFox was chosen as the main browsing environment for this research. However, any browser based on the Gecko engine should work just as well. The DOM API for Gecko engine is freely available on the Internet, which also largely improved the development progress.

Development cycle identification

Development needs to be divided into different phases where each phase has its main focus. Three development phases were identified for this research; the focus of each phase is based on the results of evaluation from the previous phase. Each phase consists of its own design and implementation stages – the product from each phase is a prototype which is evaluated using different evaluation methods.

Two types of prototyping methods are used in this research; vertical prototypes and horizontal prototypes [9]. Horizontal prototypes display a wide range of features that are close to a complete system, but with no full implementation of them. Vertical prototypes are the opposite; they do not show what a complete system may have in features, but focus on a small set of features in a near-complete fashion. Since there are three development phases, the approach here is to first set up a rough framework of the system in phase 1 (horizontal prototype); then in phase 2, a small set of features is selected (based on the results of phase 1 evaluation) and implemented. Lastly, in phase 3, the system as a whole is implemented to satisfy the aim of the research as a proof of concept. Table 3.1 lists the specifications of each phase.

Phase	Specifications
1	System design, dynamic configuration of Web services and interface
2	Improve interface configuration
3	Final system

Table 3.1: Development phases

The process of each development phase will be addressed in the following sections and the details of implementation of each phase will be discussed in the following chapters.

3.2 Phase 1

In this phase, the fundamental structures of the system, which include the system architecture, Web services architecture (provided by the digital library package) and communication component architecture, were designed. The result of this phase is a prototype that is capable of doing dynamic configuration of Web services and basic interactive user interface design.

After the first prototype was produced, a *participatory design session* was conducted, where some users were invited to look at the prototype and make alterations to the current design. The results from this session were then used as design guidelines for the development of the following phase.

3.2.1 Problems

Following are the problems identified for this phase:

- **System architecture**
  - What is the components structure?
  - The XML schema for configuration files
- **Communication component architecture**
  - A communication protocol must be defined
  - Requests/Responses processing schema (in between the System and Web services) must be defined
- **Web services architecture**
  - Identify core digital library functions to use in the system
  - Tweaking ODL components for the purposes of this research
  - Identify user customisable parameters for Web services
- **Interface functionality**
  - Dynamic Web services configuration
  - Basic page configuration
  - Drag and Drop functionality

3.2.2 Methodology

System architecture

The system adopts an architecture called *three-tier client/server* model, where the system user (using a Web browser) acts as the client tier, the system itself acts as the middle tier and the back-end digital library, which provides Web services, acts as the server-tier. Figure 4.1 in the next chapter shows this structure. However, it is possible that middle tier and server tier both sit in the same machine.

The system consists of three main components, namely communication, Web services configuration and user interface. The communication component, called the *Conveyor Module* in this project, is responsible for all the communication between the system and the Web services. All requests and responses are first processed by the conveyor module before being sent across to the other tier. The Web services configuration component is responsible for setting Web service related parameters. The UI component is the main component where user interaction takes place.

Communication component architecture

The protocol for communication between the system and Web services is defined in this component. This component takes requests from the UI component, then converts requests to an understandable format by Web services before sending them across. The responses from Web services are processed similarly in this component before returning back to the system. This is necessary in order to achieve abstraction; the ultimate goal is a seamless connection between the system and different Web services from different digital libraries.

When a request is sent to ODL components (Web services), the response is returned in the form of XML, where metadata in the format of *Dublin Core* is contained within. Dublin Core [19] is currently a prominent standard for metadata in digital libraries. Before the results can be returned and used by the UI component, some transformation must take place because data must be transformed to a more readable format than XML and the user may not require all data returned.

Web services architecture

After looking at several digital libraries, e.g. EPrints, GSDL, ODL and DSpace, two core services provided by each of these packages were identified, namely search and browse. These two functionalities provide a basic means of finding information from digital libraries; thus it was necessary to include them as core functionalities for the system. Some modification of both ODL search and browse components was necessary so that they fit into the system better and give room for configuration.

Some parameters were also identified for search and browse functionalities: e.g. number of results to be displayed per page, what subset of metadata to be displayed for results, what type of search/browse to enable, etc. These parameters complement the Web services and allow more customisation.

Interface functionality

This is the actual component where the user interaction occurs. Two sets of interfaces were designed: Web services configuration and interface design configuration.

Web services configuration allows users to customise the behaviour of Web services by setting parameters identified above in the *Web services architecture* section. Interface design pages, depending on the parameters set for Web services, will render differently to allow the user to design the basic look of the actual pages. There are two interface design pages; one for the search archive and the other for the browse archive. The basic layout of both pages is fixed; the user can only drag and drop form elements inside the page to decide how they are placed in the page. Drag and drop functionality was necessary as it allows an intuitive way of manipulating placement of elements.

Participatory design session

Three postgraduate students with a background in digital libraries from the Department of Computer Science at UCT were invited to participate in the design session. The session started with a brief introduction to the research, so participants had a broad idea of the aim of the proposed system. A short demonstration of the prototype then followed and participants were allowed to work on the prototype individually. A discussion with the participants afterward resulted in the following:

- Degree of configuration of interface was too limited
- A defined working area for designing entire page layout would be good
- Toolbars for adding contents
- Predefined template would be handy
- Google style of display of search results should be considered
- Application of styles to search/browse results is desired

3.2.3 Results analysis

The results of the participatory design session showed that more freedom in the interface design is desired. These freedoms include dynamic manipulation of page styles (look and feel) and the inclusion of additional content. Most of the comments focussed on the interface functionality instead of the system itself. Thus we claim that the design of the underlying system is sufficient at this point and the focus of the next phase of development should concentrate on the usability of interface.

3.3 Phase 2

The main focus of this development phase is to improve the interface design functionality. Based on the results obtained from the previous phase, it was found that the direction is heading towards implementing a Web page editor inside a Web browser. Thus some of the most popular Web page editors, such as Macromedia DreamWeaver and Adobe GoLive, were studied. Together with the results of the design session, some design concepts for this phase were identified.

Expert evaluation was conducted after the completion of the second prototype. This evaluation involved two participants from the Department of Computer Science at UCT who are specialised in the field of digital libraries and interface design. The results from the evaluation were then used as design guidelines for the next development phase.

3.3.1 Problems

The following are the problems identified in this phase of development:

- A defined layout area which represents the page to be designed
- Toolbar/panel must be designed
- Following properties for layout area identified
  - Background images/colour
- Insertion of following types of element identified
  - Image
  - Text
- Insertion of Web services fields
- Uploading images to the system

3.3.2 Methodology

Layout area for interface design

What is needed here is a visual representation of the page to be designed. Similar designs can be found in most *What You See Is What You Get* (WYSIWYG) type Web page editors, such as Dreamweaver or GoLive, where there is a design area which represents the page being created and a toolbar running along its side. This setup is followed in this project where a fixed size block sits in the middle (representing the page) and a toolbar sits on its right. Figure 5.2 shows the structure.

The toolbar contains all available elements that can be inserted into the defined layout area, as well as a page properties setter, such as background image/colour and page size. The size of the toolbar can be lengthy, thus it is divided into different tabs where each tab can be expanded or collapsed.

In this phase, there are two pages with this functionality implemented, namely the search interface design page and browse interface design page. Thus, the user is allowed to design the look and feel of both Web services pages. These two pages will be referred as *interface design pages* in this document.

Insertion of elements

There are two main element types as discussed above, namely text and image; a new tab in the toolbar is created for each element type. The text element tab contains two subtypes of text element, single line text and multi-line text. Image tab displays a list of clickable thumbnails of all available images in the system. The collection of images can be expanded by uploading new images to the system.

Insertion of Web services fields

In Phase 1, Web service elements could not be dynamically inserted; the user needs to first define the number and types of field in the Web services configuration page and then design their placement in the interface design page.

In this phase, we want to allow insertion of Web services fields at design time, so the option of defining a number of Web services fields in Web services configuration page is removed and a tab in the toolbar of interface design page is created. This tab contains either a search field or browse field, depending on what Web services page is being created. A search field consists of two form elements; a dropdown list and a text field. The dropdown list is a list of Dublin Core elements and the text field is for entering search phrases. There are various types of browse field; depending on what sorting methods the Web services support, different browse fields will be displayed in the tab.

Uploading images

This functionality is needed to allow user to upload their own images, so these images can be used when creating pages. A folder is created on the server to store these images. This folder is read by the image tab in the toolbar of the interface design page to create a list of thumbnails of all available images in the folder.

Expert Evaluation

Two participants from Department of Computer Science of UCT, who specialise in HCI and digital libraries were invited to do the evaluation after the completion of the second prototype. A brief introduction and demonstration of the system was given before the participants worked on the system themselves. A focus group with the participants

was held afterwards; a few problems were identified and some possible solutions to the problems were discussed. The following are the findings from the evaluation:

- User should be allowed to define a workflow for the pages created
- Lower level of granularity on text elements is desired

3.3.3 Results analysis

The interface design page at this stage is still a single page configuration; the structure of Web site and the relationships amongst pages cannot be defined. It was suggested to design a function which allows users to create an arbitrary number of pages and define their relationships. The prototype at this stage supports no styling of text inserted into the layout area – it was pointed out that some styling of text will increase the readability of pages created.

The addition of the suggested features can lead to more problems; by allowing the user to create a site structure, the system must be able to create a structure containing an arbitrary number of pages together with their relationships and their design. The pages must then be rendered. A navigation system must also be designed so the user can move within the site without problems. Thus these are the problems to be solved for the next phase of development.

3.4 Phase 3

The focus of this phase is to consider the findings from the evaluation of the last phase and create a working system. With the feature of creating site structure and navigation, the functionality provided by the system will be similar to what is provided by most Web page editors.

The same experts who were invited in phase 2 for evaluation were invited again for the final evaluation. They examined the final system to see if the system is satisfactory and meets the expectations from the evaluation of phase 2.

3.4.1 Problems

From the results of the analysis from the last development phase, the following problems are identified:

- Site structure creation (pages and relationship) must be designed
- Design/render for arbitrary number of pages
- Navigation system for the final site must be designed

3.4.2 Methodology

Site Structure

To visually aid the user to create a site structure, the challenge here is to dynamically render the structure as the user adds more pages. The standard solution is to use a tree-like structure to represent the site; the advantage of using a tree structure is that the relationships between pages are defined when new elements (pages) are added to the structure.

A few page types are predefined so users can choose what type of page to add to the site. These page types are Home, Help, Blank, Search archive and Browse archive. To create a site, the user first needs to select a page type and add it as the root, then a site map of the site structure is rendered dynamically to assist the user to see the site visually. Users can expand the site by selecting a page type and an element in the site map to add a child page. Figure 6.1 and 6.2 demonstrate how this is done.

Design/render arbitrary pages

The functionality of the interface design page needs not be changed, but the editing type of interface design page must be changed to a generic one, because in the last phase, only search and browse interface design pages were implemented with the system. If the interface design page is generic, it can be used to design any arbitrary page in the site created by the user.

Interface design for Web service pages was split into two sections in this phase; a main page and a results page. This idea came from the design of Google; it was suggested in

the evaluation results of phase 1 to consider the Google style of results display. With this dual page mode, users can design both pages to create a more intuitive set of interfaces for Web services.

Once a page is designed, the settings are saved to the configuration file (in the form of XML). These settings are retrieved again when a particular page needs to be rendered. Thus settings for each page must be stored under a unique id so they can be uniquely retrieved.

Navigation system

A navigation system is required in order to build a menu for each page, so that a user can navigate within the site. There are two types of general link: *Home* link which links back to root page and *Child* link which links to all children pages. The user is allowed to define what type of links should be included in the navigation system, as well as the style (look and feel). In this way, the navigation menu in all the pages will share the same style.

Experts Evaluation

The same experts who were invited in the evaluation of phase 2 were invited again to examine the output of this phase. The purpose of this is to allow these experts to compare the system to the suggestions from the last phase and see if the system is satisfactorily implemented.

This evaluation started with a brief introduction to the new features and changes of the system from the last phase. This time, a task list was provided and participants were asked to complete these tasks using the system to ensure they explored the functionality more fully. Observations were made during the process for the purpose of results analysis. Each participant was interviewed shortly after they completed the tasks and their thoughts and suggestions were noted.

The following are some points concluded from the evaluation:

- The current notion of how site structure is created, where the relationship between pages is defined as new pages are added to the structure, may not be

intuitive enough. It was suggested that first creating all the pages, then defining the relationships amongst them would be a preferred option.

- A small view of the site structure while in the process of designing pages will help the user in building a mental model of where they are in the process.
- The definition of styles on the navigation system is too limited
- The option to preview a page when done designing is required
- Alignment of elements in interface design page is difficult
- Some predefined template would help to speed up the process of interface design

3.4.3 Results analysis

From the observation, it was found that sometimes participants got lost in the process of creating a site structure and sometimes experience difficulties in interface design pages. This is most likely caused by the explanations in the pages themselves not being clear enough. One participant said, "All the basic functionality for creating an interface system is there, but the system may need to reshuffle the order in a more sensible way."

The findings from the evaluation were mostly focused on improving features and the usability of the system, but not much comments on request for system functionalities. It is assumed that the system developed is a feasible one and necessary functionalities are implemented. Improvements to the functionalities of the current system will be left for future work.

Chapter 4

Phase 1

This phase is the initial stage of the development where the actual implementation started. This chapter describes how the initial environment for the system was set up and the details of the implementation of the initial system.

4.1 Setting up Open Digital Libraries components

The digital library back-end was set up before any actual implementation of system took place. As described in the *Development cycles* chapter, Open Digital Libraries was the digital library software chosen for this project. The two core Web services, search and browse, are used.

Two ODL components were used to establish search and browse Web services. IRDB v1.3 [15] is the ODL component that provides the search Web service. It harvests metadata from some repositories and stores indices for them in a database. The default database system used by IRDB is MySQL, which is a very popular open source database system. DBBrowse v1.2 [15] is the browse Web services component of ODL; like IRDB, DBBrowse also harvests metadata from some repositories and stores them in a database. The repository used for metadata harvesting is *Hussein's picture album* [30], which contains descriptions of collections of photos.

These two components were installed and set up to run in a Linux Web server and MySQL database. Both components were configured to harvest from the repository, using the default metadata format, *Dublin core*. DBBrowse was specifically configured

to index for browsing on a few fields and filter on date field. These Web services can be requested by HTTP requests with some parameters; the URL for making requests are called *BaseURL*.

4.2 System architecture

As described in the development cycle chapter, this phase is to establish a basic framework for the system. This framework includes a Web interface for the system, communication and configuration of Web services (conveyor module) and basic interface design functionality. Figure 4.1 illustrates the structure.

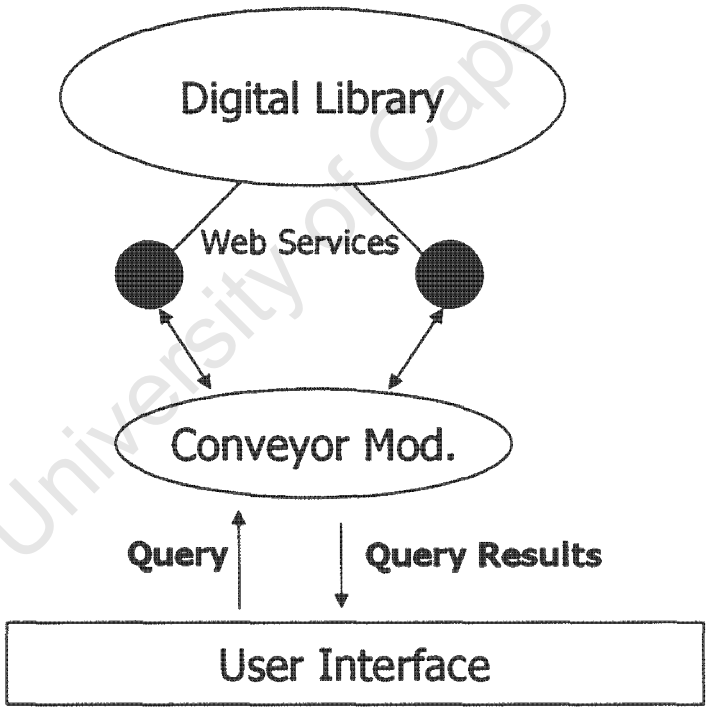

Figure 4.1: Phase 1: System architecture

From Figure 4.1, we see that the *conveyor module* is the key component between the digital library and the system, which is responsible for processing requests and results. The implementation of the conveyor module will be discussed in detail in the next section. Since PHP5 does support the concepts of object and classes, object orientation is upheld wherever possible in the implementation of the system. The conveyor module

itself is an object class, while the user interface consists of some PHP, CSS and XML files. All settings that need to be stored for future usage are recorded in some XML file. PHP5's DOM parser is also used to parse XML when data needs to be retrieved.

4.3 Conveyor module

This is the core component for communication between the digital library and the system. The design goal here is to strive for maximum flexibility so that if the system is to be used with other digital libraries, this module can be replaced with a compatible one.

As mentioned in the previous section, this module is an object class itself, thus a new instance of this is created in order to establish communication with Web services. Two parameters are required for the constructor, namely *XSLPath* and *Number of entries per page*; both of these parameters are used for XML transformation (this will be discussed later in the section). When a new instance is created, the base URLs of the ODL Web services are loaded from an XML file, so that communication with search and browse Web services is possible.

When calling methods which establish communications with Web services, a query string and paging information must be supplied. Thus the full HTTP request can be prepared and sent to corresponding Web services. Requests are sent by using *fopen* in PHP5, which establishes an HTTP connection to the URL which was supplied as a parameter. Results returned by Web services are then retrieved by using *fread*. Illustration 4.1 shows a piece of XML results from ODL Web services.

Figure 4.2 illustrates how results are processed in the conveyor module. When results are retrieved from ODL Web services, they are in the form of XML; they must be transformed before the user can make sense out of them. Thus in the conveyor module, after results are retrieved, they are loaded into an XML DOM and then an XSL file is used to perform XML transformation. Hence, the transformed results are sent to the interface to be displayed. The XSL file used here is dynamically generated when users configure the Web services. This will be discussed in detail in the next section.

```
<?xml version="1.0" encoding="UTF-8"?>
  <xoai:ListRecords xmlns="http://www.openarchives.org/OAI/1.1/OAI_ListRecords"
      xmlns:xoai="http://oai.dlib.vt.edu/XOAI/1.0/XOAI_ListRecords"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.openarchives.org/OAI/1.1/OAI_ListRecords
          http://www.openarchives.org/OAI/1.1/OAI_ListRecords.xsd
          http://oai.dlib.vt.edu/XOAI/1.0/XOAI_ListRecords
          http://oai.dlib.vt.edu/XOAI/1.0/XOAI_ListRecords.xsd">
    <responseDate>2005-12-13T22:51:01+02:00</responseDate>
    <requestURL>http://simba.cs.uct.ac.za:80/~ffeng/cgi-bin/ODL-IRDB-1.3/IRDB/myDL/
      search.pl?verb=ListRecords&amp;set=odlsearch1/+creator:Hussein/1/10
      &amp;metadataPrefix=oai_dc</requestURL>

    <record>
      <header>
        <identifier>oai:hspics:1997021ba4</identifier>
        <datestamp>1997-12-16</datestamp>
        <setSpec>1997021ba</setSpec>
      </header>
      <metadata>
        <oaidc:dc xmlns="http://purl.org/dc/elements/1.1/"
            xmlns:oaidc="http://www.openarchives.org/OAI/2.0/oai_dc/"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/oai_dc/
                http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
          <title>971ba4</title>
          <creator>Hussein Suleman</creator>
          <subject>The airport</subject>
          <description>Luc, Sajida, Ne, Fuzel, Hansel - u guys never expected these
              photos to become public on the internet (hehehe)</description>
          <publisher>Hussein Suleman</publisher>
          <date>1997-08-05</date>
          <type>image</type>
          <format>image/jpeg</format>
          <identifier>http://www.husseinsspace.com/pictures/1997021ba/971ba4.jpg</identifier>
          <language>en-za</language>
          <relation>http://www.husseinsspace.com</relation>
          <rights>unrestricted</rights>
        </oaidc:dc>
      </metadata>
    </record>

  ...

    <xoai:responseContainer>
      <hits xmlns="http://oai.dlib.vt.edu/ODL/Search/1.0/responsecontainer"
          xsi:schemaLocation="http://oai.dlib.vt.edu/ODL/Search/1.0/responsecontainer
          http://oai.dlib.vt.edu/ODL/Search/1.0/responsecontainer.xsd">30</hits>
    </xoai:responseContainer>
  </xoai:ListRecords>
```


Illustration 4.1: ODL IRDB search: raw results sample

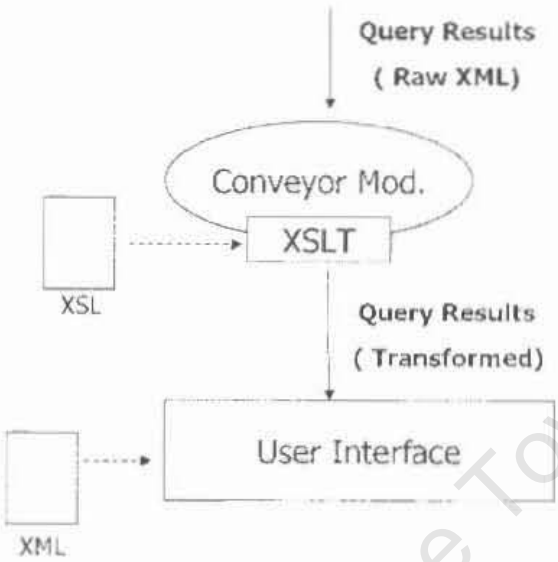

Figure 4.2: Conveyor module results processing

4.4 Web services configuration

A few Web service properties were identified in the development cycle chapter as needing to be configured by the user. Thus a configuration page for each Web service is implemented.

4.4.1 Configuration interface

Figure 4.4 is a screenshot of the configuration page for the browse service. From the screenshot, two categories of configuration can be seen, *sorting category* and *filter category*. Sorting category allows the user to choose preferred options to browse the archive according to certain fields; the filter category allows user to enable or not the date filter. These options in both categories are generated dynamically; this means, the supported functionality is first requested from the Web services before being rendered to the configuration interface.

Metadata display allows the user to select the metadata to be rendered when displaying request results. *Navigation position* lets users decide where in the page the paging information is to be rendered, (or not to be rendered at all). *Number of entries per page* provides finer configuration on paging of data.

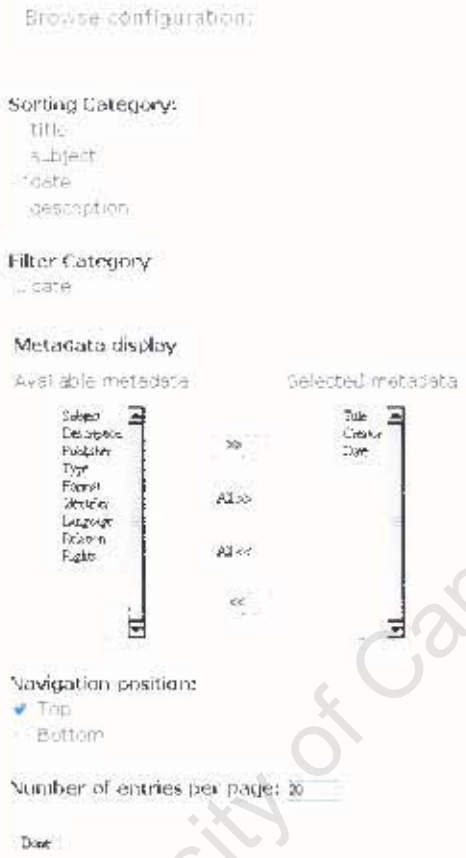

Figure 4.3: Browse configuration

Figure 4.4 is a screenshot of the configuration page for the search service. It is similar to the browse configuration page, but with a few differences. Instead of the two categories of browsing options, search configuration lets users choose what type of search mode to use. *Simple search* is similar to any current search engine where one text box is shown to let users enter keywords. *Field search* is a more advanced search mode where users can search for some keywords against certain metadata fields (from Dublin core). Field search can be stacked, so more than one matching criteria is allowed.

An object class called *create_config* was created to take care of saving settings. In the case of Web service configuration, after the HTML form is submitted, an instance of create_config is instantiated in the post process page. All settings are passed into a method as parameters before they are saved in the configuration XML file. Once these settings are saved to the configuration XML file, the last bit of the remaining

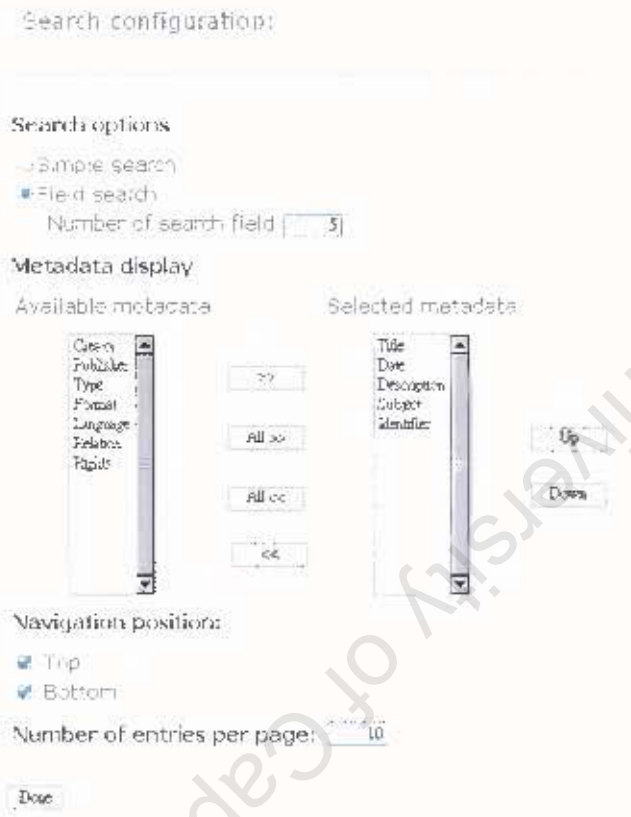

Figure 4.4: Search configuration

process is to do an XML transformation to generate a XSL file, which can be used when transforming request results.

4.4.2 Dynamic XSL generation

From illustration 4.1, each record in the results returned by ODL Web services contains a full set of Dublin Core metadata, but the user may only want to display a subset of it; thus an XML transformation is required. To do the transformation, an XSL file must be presented which transforms the returned results into a desired format. This XSL file must be dynamically generated by the system, depending on what Dublin Core fields the user chose to be displayed.

Since XSL is specified in XML, it is possible to generate XSL from an XML transformation. An XSL file was written such that it is capable of generating XSL tags and

code. Thus when we run this XSL to transform the configuration XML (after new settings were saved), the generated XSL file will do the transformation as specified by the user. Figure 4.5 is a more detailed version from Figure 4.2, and illustrates the relationship between components. The generated XSL is used by the conveyor module to transform the received results to the format specified by the user.

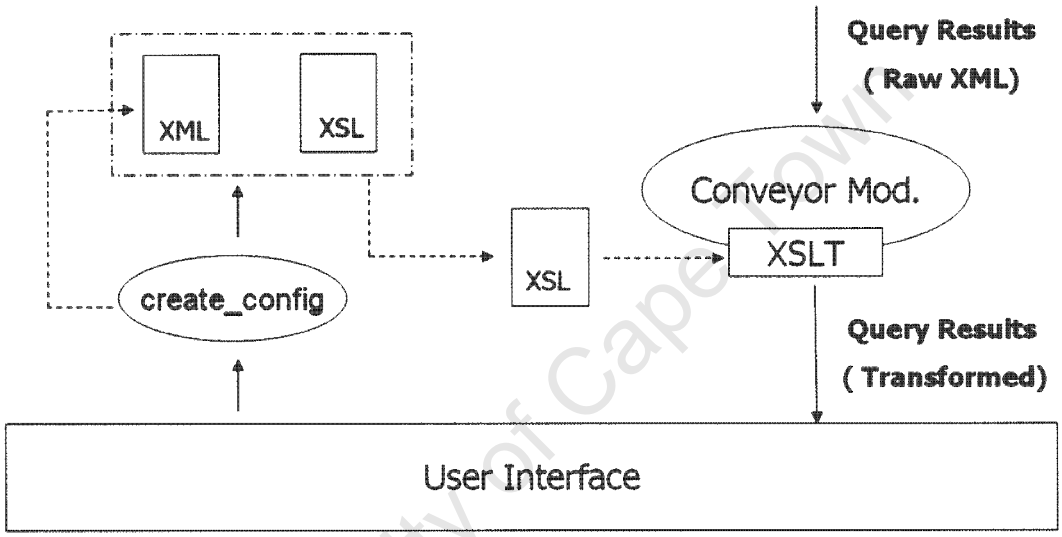

Figure 4.5: Dynamic XSL generation

4.5 Web services interface design page

After the user has configured Web services, the next step is to configure interfaces for Web services. In the previous section, the user defined the mode for browse/search; depending on what was configured, corresponding elements are rendered and allowed to be dragged and dropped in the interface design pages. Figure 4.6 is a screenshot of the rendered interface design page for the browse service.

In the screenshot, three elements are rendered and they can be dragged and dropped anywhere in the page. The drag and drop functionality is made possible by using open source JavaScript libraries available on the Web – *DOM-Drag*, developed by Aaron Boodman, and *Walterzorn Drag and Drop* developed by Walter Zorn.

Figure 4.6: Browse interface design page

DOM-Drag was initially used in the project. The idea is to make an element become drag-able and use two hidden text fields to record the X-Y coordinates of the element. On the end of each dragging event, a JavaScript function that updates the two text fields is executed to update the new X-Y coordinates of the element. When the form is submitted, these text fields are submitted together to be processed. Illustration 4.2 is an example of how this is done.

Part A of the example makes an element drag-able and specifies the onDragEnd event to trigger the function *reportDragEnd* (Part B). Part B is the JavaScript function which updates the value of two text fields with the coordinates of the element being dragged. Part C is the declaration of the actual HTML element and Part D is the two text fields, which are set to be hidden. Since it is known what elements are to be rendered for interface design pages, these text fields are generated by PHP scripts at compile time as the page gets executed. The JavaScript code (Part A) to make elements drag-able is standard and is generated at compile time for each element to be made drag-able; the problem of doing this is that if the number of elements becomes large, the maintenance of code becomes more tedious.

When the user is done placing all elements and clicks on *Done* to end the design process,

```
<html>

    <script language="javascript">

    ...

    //Part A
    var element = document.all ? document.all["myElement"] : document.getElementById("myElement");
    Drag.init(element);
    element.onDragEnd = function(x, y) { reportDragEnd("myElement", x, y); };

    //Part B
    function reportDragEnd(id, x, y)
    {
        var obj_x = document.all ? document.all[id + '_xpos'] : document.getElementById(id + '_xpos');
        var obj_y = document.all ? document.all[id + '_ypos'] : document.getElementById(id + '_ypos');
        obj_x.value = x;
        obj_y.value = y;
    }

    ...

    </script>

    ...

    <!-- Part C -->
    <img src='block.jpg' id='myElement' style='position:absolute; top:50px; left:50px;'>

    <!-- Part D -->
    <input type='text' id='myElement_xpos' name='myElement_xpos' value='0' style='display:none;'/>
    <input type='text' id='myElement_ypos' name='myElement_ypos' value='0' style='display:none;'/>

    ...

</html>
```

Illustration 4.2: DOM-Drag sample code

coordinates of all elements are saved to the configuration XML file so they can be used when rendering the actual page.

4.6 Interface Rendering

To render the designed pages, absolute positioning from CSS is used to place elements at their specified positions. Since coordinates of elements are saved in the configuration XML file, it is easy to retrieve them by parsing the XML file in PHP. Once these coordinates are retrieved, rendering of elements is easy by using PHP scripts. The following is a section of the code of the rendering process and a screenshot of the rendered browse service interface.

```
<!-- Part A -->
<div style='position:absolute; left:267px; top:192px'>

<!-- Part B -->
<input type='checkbox' name='sort_title' value='1' onClick='enable(order_sort)'> Title

<!-- Part C -->
<select name = 'order_title' disabled>
    <option value='+'>Ascendent</option>
    <option value='-'>Descendent</option>
</select>
</div>
```

Illustration 4.3: Interface rendering sample code

Illustration 4.3 is the section of the code which generates the encircled element in Figure 4.7. In part A of the illustration, the coordinate information is generated by PHP scripts by retrieving them from the configuration XML. The same is true for element names in part B and C. By looping through all the recorded elements in the configuration XML file, the interface designed by the user is re-created.

When a request is sent and results are retrieved (from the conveyor module), the same interface page is rendered together with the paging information and retrieved results directly underneath. Figure 4.8 is a screenshot of the browse results page.

Figure 4.7: Browse Interface

Figure 4.8: Browse results page

4.7 Phase conclusion

4.7.1 What was found in this phase

In the "Phase 1" section of the development cycle chapter, the results from the evaluation were discussed and it was concluded that the interface design page should be

more focused and improve on functionality. Instead of predefining which browse/search elements to have, adding of elements at run time would give a more intuitive design. Users should also be able to design the look and feel of the entire page layout instead of merely deciding the placement of browse/search elements.

It was also found that there was a lot of repetition of code in the development of browse and search services. This is largely due to the similarity in how both their functionalities and interfaces were designed to be configured in the system. The dynamic generation of JavaScript code was also found to be very tedious; since JavaScript cannot be mixed with PHP Scripts, all JavaScript related to making elements drag-able must be dynamically generated. This structure of code generation is hard to maintain by its nature and may cause problems in later stages of the project.

4.7.2 Influence on design

It was decided to carry on to the next phase with just the development of one Web service, since the implementations of the two services are very similar. The idea of allowing the user to design just a portion of the Web service interfaces was also changed, since the results of evaluation show that the user desires more freedom in interface design – it is better to allow users to design a complete page from scratch.

Chapter 5

Phase 2

This development phase focuses on the structure of the interface design, thus requiring a vertical approach to implementation. The design of the system in this phase is based on the results of evaluation from phase 1. New JavaScript libraries were used and some improvements were also made to the system. Together with the implementation of the new structure of the interface the design page, these are all discussed in this chapter.

It was decided to keep only the search service for the development of the system. However, during the course of implementation, browse service related functions were reserved (with empty function headers) for future implementation.

5.1 Improvements

A few improvements were made to the Web services configuration. One of the main alterations was the visualization of metadata configuration. In the original design, the user had to select the metadata elements which are to be rendered in the results page by multiple clicking in order to add them to the collection. Please refer to the screenshot in Figure 4.4.

The idea of having two lists and letting users select the display set of metadata was found not to be intuitive. It is a more intuitive design if the user is allowed to select and drag preferred elements. The user first selects the set of preferred metadata for results display and then drags the selected elements to change their display order.

In Figure 5.1, we can see the configuration for search modes was taken away and the

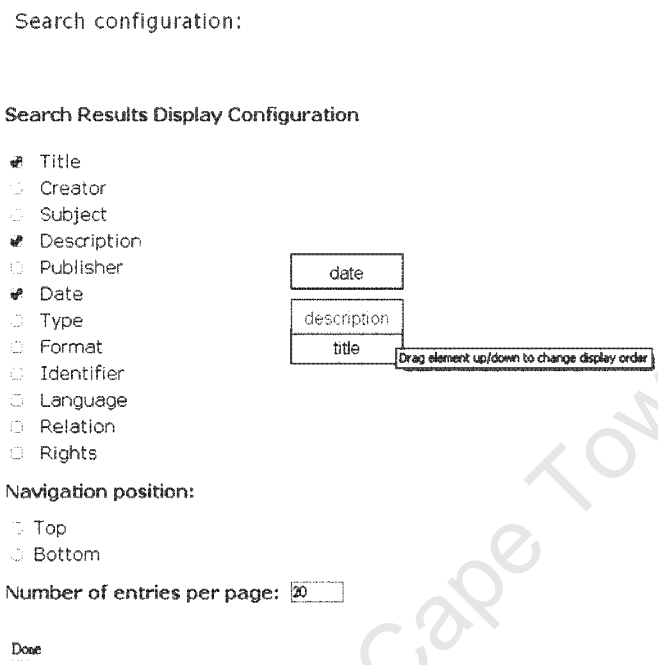

Figure 5.1: Improved search configuration

structure for configuring metadata has changed from two lists of items to a list of checkboxes and a drag-able visualisation of the selected elements. The user chooses the metadata set by *checking* items on the left and the system will dynamically generate the selected metadata items on the right. In other words, the visual rendering on the right reflects what is chosen on the left.

This dragging structure of a list is enabled by using a JavaScript library called *Drag&Drop Sortable Lists* by Tim Taylor Consulting. This library enables sorting of a list of items by drag and drop, which is done purely by JavaScript and CSS. When a user checks or unchecks an item, a JavaScript function is triggered to re-render the sortable list on the right. After the user is done with ordering of the list and submits the form, another JavaScript function is triggered to record the list order in a invisible text field before the form is submitted, so that this list order can be processed after form submission.

5.2 Interface design page

The idea here is to have a visualisation of a page (layout area) being built and providing a toolbar of different elements that can be inserted into the layout area. This structure is similar to most Web editors, the major difference is that the system being built here is based on a Web browser, thus the development framework is more limited.

5.2.1 Change of JavaScript library

A new JavaScript library was found to be suitable for the purpose of this research, called *Walterzorn Drag and Drop library*. This library is flexible in terms of development and is also easy to use. Many parameters can be applied when making an element drag-able, e.g. only horizontal/vertical movement, borders, transparency, etc. Another advantage of this library over the one used in phase 1 is that fewer procedures need to be defined to make an element drag-able while more features can be specified at the same time. This is especially advantageous for error checking and code maintenance.

In this phase, we are allowing users to insert elements and make them drag-able at run time instead of at compile time (this will be discussed in detail later in the chapter), thus no JavaScript code needs to be pre-generated by PHP scripts, but all processes are done at run time by JavaScript.

The author of the library also allowed some extensibility to the library; four empty functions were allowed to be extended by library users. These four functions are *my_PickFunc*, *my_DragFunc*, *my_ResizeFunc* and *my_DropFunc*, and are triggered when an element is picked, dragged, resized or dropped respectively. This is particularly useful to this project and some of these functions are extended to do some monitoring work. More will be discussed in the following sections.

5.2.2 Layout area

This area is the place representing the page being designed. All elements are inserted into this area and styles can be applied to them. Figure 5.2 shows the look of the interface design page. The block on the left is the layout area and the tabs on the right constitute the toolbar. The block is a fixed size HTML *td* element, and when elements are inserted, they are inserted between the <td> and </td> tags as children

elements. The toolbar tabs can be expanded and collapsed, so that the user can choose what parts of the toolbar are to be seen.

Figure 5.2: Interface design page

Toolbar

A few categories of tools were implemented and made into tab structures. Each tab can be expanded, as well as collapsed; this is done by hiding the contents of the tab through JavaScript and CSS. Each tab in the toolbar contains either elements that are clickable (insertable) or relevant settings for the layout area. When a clickable element is clicked, the element is inserted into the layout area. Five categories of tools were implemented. The following explains their purposes and Figure 5.3 shows how they look when these tabs are expanded.

- **Layout properties** This tool tab is for changing the size of the layout area in terms of screen pixels.
- **Web services components** There are two types of Web service components, browse and search; only the search component was implemented. The search component consists of a dropdown list and a text box encapsulated in a *div* tag.

Figure 5.3: Toolbar

- **Images** All the images uploaded are stored in a folder on server, these images are displayed here in thumbnails. The following image formats are supported: jpg, gif, png and bmp.
- **Texts** Two types of text components were implemented; text and paragraph. Text is for inserting single line text and paragraph can be used to enter multi-line texts.
- **Background properties**. A dropdown list of colours can be used to set the background colour of the layout area. The same images as in the image tab are displayed here in thumbnails as well. A button is also provided to remove background images of the layout area.

When an element is clicked or a setting is set, a corresponding JavaScript function is triggered to perform some actions. This interaction is discussed in detail in the following subsection.

5.2.3 Layout area - Toolbar interaction

As mentioned earlier in the chapter, we have shifted the elements insertion from compile time to run time, which means all the interface manipulation work must be performed in JavaScript instead of PHP scripts. This is where DOM is indispensable for the implementation of the project.

With the aid of DOM, it is possible to access any element in the page if a unique ID is assigned to every element; it is also possible to add, modify and delete any elements in the page. Thus the idea here is when a user clicks on an element in the toolbar, a JavaScript function is triggered which makes a copy of the element that has been clicked. The same is true for setting properties of a layout area; by acquiring the handle of the layout area (td element), it is possible to change the properties of it, such as size, background colour or background image.

Insertion of elements

When inserting an element, other than cloning the element from the toolbar, a few other elements need to be created too. Two text fields must be created to record the X-Y coordinates of the element; an extra text field is needed to record the file source if the element is an image. There are three special text fields that are associated with the three types of element: Web services, text and image. These text fields keep count of the number of each type of element inserted into the layout area, so when the form is submitted, the post process can process this data accordingly.

The cloned element must also be given an ID, so that it can be uniquely identified and made drag-able. These ID's are generated by the concatenation of *element type* + *counter*.

Deletion of elements

The system also allows users to remove elements from the layout area. This is done by registering an event handler with an element. The *Double-click* event is used in this project to trigger the remove function; this remove function not only deletes the element itself, but also does all the cleaning and removing of all associated text fields. Illustration 5.1 demonstrates how this is done.

```
clone.addEventListener("dblclick", remove, false);

...

function remove(e)
{
    //Remove element from layout area

    //Remove position text fields of the element

    //If current element is image, delete image src text field

    //If current element is text, delete text contents text field

}
```

Illustration 5.1: Removal of element

Note that the remove function does not decrement the counter text fields, because this may affect the generation of unique ID's and introduce problems in the post process of data collection. The checking of validity of data is left to the data collection procedure in post-process.

Drag and drop of elements

When elements are inserted into the layout area, they are also made drag and dropable. Since we are inserting elements into the layout area, the problem here is that these elements should not be dragged outside the border of the layout area. This functionality is available in *Walterzorn Drag and Drop library*, when an element is made to drag and drop, it can be defined how far the element can be moved away in all four directions from it's original position when it was inserted.

The calculation of the four offset distances are demonstrated in the following table:

From Figure 5.4, it is easy to understand how these equations are derived. The X-Y coordinate of an element is referring to the coordinate of top-left corner of the element.

Lastly, the function *my_DropFunc* of Walterzorn Drag and Drop library is extended to update the two coordinate text fields. Thus, when an element is dragged and dropped, this function will be triggered and update the two text fields with the latest coordinates. It is important to remember here that we cannot just record the coordinates of the

Direction	Offset distance calculation
Right	Layout width - Element width - (Element X coordinate - Layout X coordinate)
Bottom	Layout height - Element height - (Element Y coordinate - Layout Y coordinate)
Left	Element X coordinate - Layout X coordinate
Top	Element Y coordinate - Layout Y coordinate

Table 5.1: Offset distances calculation

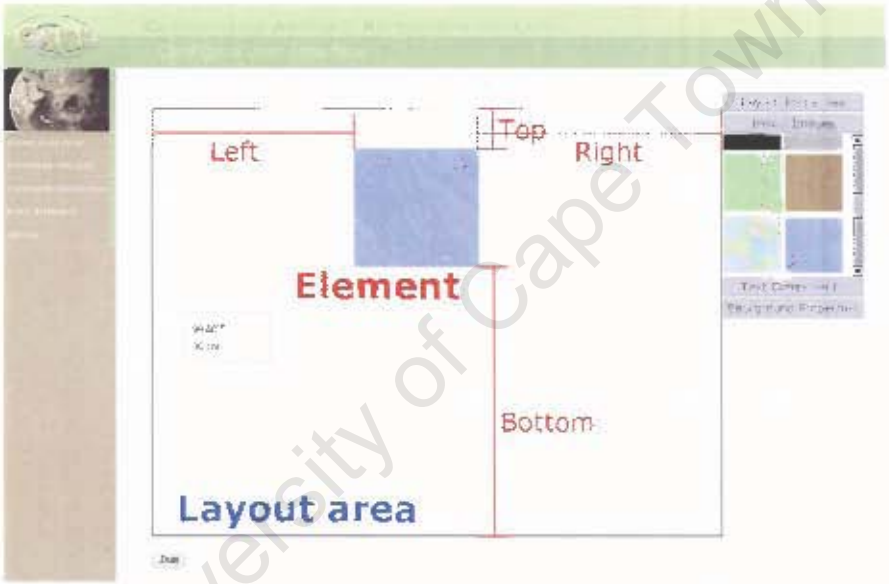

Figure 5.4: Offset distances calculation

current position of the element, but have to calculate how far the element is away from the top and left border of layout area, so when we are rendering the actual page, elements can be correctly placed.

5.3 Upload

This function is to allow the user to upload images that can be used when designing interfaces. Since the interface is being designed online, if the user wants to use some custom images, these images must be uploaded to the server before they can be used. All images uploaded are stored in the same directory and are listed as thumbnails under

the image tab of the toolbar.

5.4 Phase conclusion

5.4.1 What was found in this phase

From the evaluation of this phase, it was found that the system lacks one core ability – the definition of workflow. Since the interface design page is solely for designing Web service pages, it would be useful to allow a user to define a workflow of pages and do design work on each of those pages. Another suggestion was to implement more styles options for text, but this is an add-on to the system instead of a new feature – the implementation of it is greatly dependent on the time available for development.

The results page of Web services remained the same from phase 1, that is, the same page is rendered with all retrieved results rendered directly underneath. Such a design looks somehow awkward because all Web services elements seem to be fall out of place with all results appended at the bottom, this does not fit in the system at all.

5.4.2 Influence on design

To allow definition of a workflow, or in other words, constructing a site structure, some new subsystems may need to be designed and implemented. The very first one is a navigation system that links all pages in the site structure together, otherwise some pages may not be accessible. Another one is a site map rendering system, which can render an overview of the structure created by the user.

The Web services page should be split into two pages, where the user needs to design a separate interface for both parts (so that the interface for results page can have a different look compared to the main Web services page). This follows the Google search engine which has different appearances for the query and results pages.

Chapter 6

Phase 3

This final phase of development concentrated on making the system a complete one. The two main implementation focuses of this phase are the definition of workflow and the navigation system. With the completion of these two components, the system can be said to have achieved the aim of this research: a system that allows user to design their own interfaces for digital libraries in an intuitive way within a Web browser.

A separate interface design page and a stand-alone rendering page were implemented for the results page in this phase. This separate set of interfaces is necessary to design and render the results page, because some settings for the design of results page need to be first defined. This will be explained in detail in the following subsections.

6.1 Workflow

Two things must be defined when building a workflow (site structure): *page type* and *page relationship*. Page type is to identify a Web services page from a normal page, so that Web services-specific elements can be rendered on the interface design page; page relationship defines how a page is linked to other pages in the workflow. A few page types were identified as necessary for the system – *Home page*, *Help page*, *Blank page* and two Web services pages: *Search archive page* and *Browse archive page*.

The design here is to let users build a site structure by adding pages. The user starts with an empty site, as illustrated in Figure 6.1; by choosing a page type from the dropdown list and selecting a page in the site map rendered at the bottom, the page

Figure 6.1: New Workflow

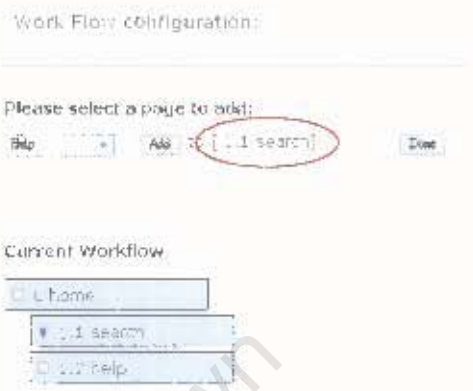

Figure 6.2: Workflow with 3 pages

can inserted as a child page to the page selected. Figure 6.2 illustrates three pages having been inserted. In both figures, the description encircled in red changes as the user selects a page in the site structure. This helps the user to see where the page is inserted.

In Figure 6.2, the site map at the bottom is rendered in real time. The following two subsections will discuss the data structure used to generate this structure and the rendering process.

6.1.1 Data structure

To generate a site structure, a data structure which can hold information for every page in the site is needed. Each page has a *level id* and a page type, together with its location within the site.

An n-ary tree structure is designed to perform the job. Each element in the tree is essentially an array of two elements, a text element and an array element – an array within an array. The text element records the level id of a page and the array element keeps track of all links to the child pages. Figure 6.3 is an example site structure and Figure 6.4 illustrates how the data structure models the example site structure.

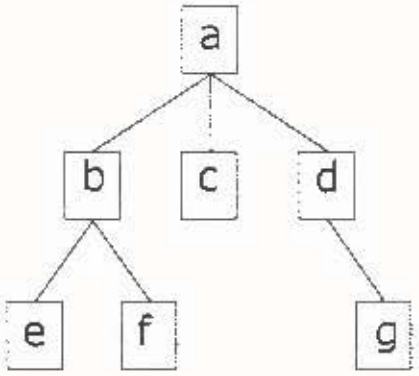

Figure 6.3: Example site structure

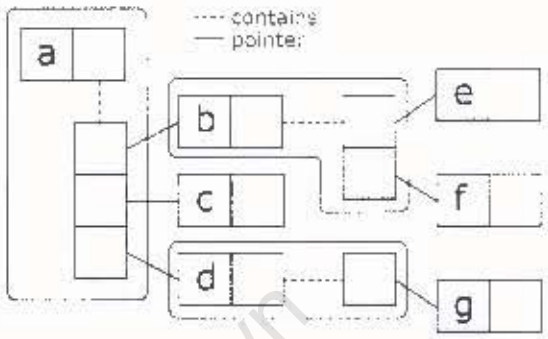

Figure 6.4: Example data structure

Note that in Figure 6.4, elements *c*, *e*, *f*, *g* are all an array of two element with the second element set to null. The second element (array) will only be initiated when the first child page is to be inserted. This is implemented to save memory space. The site structure can be divided into different levels where the root level is 0, so the level id for each page can be generated by concatenating the level of the page in the site with its page type. For example, *1.2 Search* means that the page is a search archive page and it is the second page of level 1 in the site.

To traverse the tree, a recursive traversal is required. Thus searching for a single element in the tree may end up traversing the whole tree if the element is at the outermost level. Since the size of the tree will not be huge, efficiency of the algorithm will not be an issue. When the user is done defining the site structure, the system needs to save this structure to the configuration XML. Thus this tree structure needs to be converted to XML; this is done by traversing the entire tree and building the XML as traversal gets deeper into the tree. Illustration 6.1 depicts how the XML will look when a tree is converted.

6.1.2 Site map rendering

Figure 6.3 shows a typical way of displaying a site map. However, it is difficult to render such a tree structure style in a browser, due to how HTML elements are rendered in Web browsers. To render a top-down tree structure in HTML, placement of elements in the page and drawing of lines connecting elements are rather difficult.

```
<workflow>
    <level0 pagelevel="0" pagename="home">
        <level1 pagelevel="1.1" pagename="search"/>
        <level1 pagelevel="1.2" pagename="help"/>
    </level0>
</workflow>
```

Illustration 6.1: XML structure of the tree structure

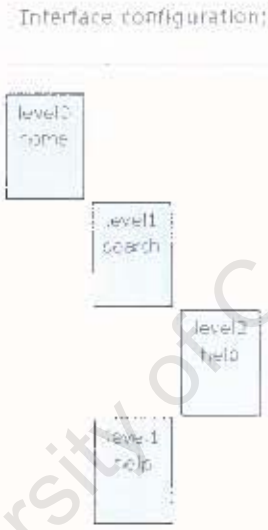

Figure 6.5: Site map

However, it is much easier to display a tree structure in a cascading fashion. Figure 6.5 illustrates how it is done in the project. Pages at different levels are aligned accordingly and are easily recognised. The site map displayed at the bottom of Figure 6.4 is rendered in the same way, but styled differently. Figure 6.5 is displayed when a user wants to build up the interfaces. Each of the pages in the site map is clickable and it takes the user to the interface design page once it is clicked.

6.2 Navigation System

Once the site structure is defined, it is necessary to establish how pages are linked together. This is done by creating a menu within every page where the menu contains

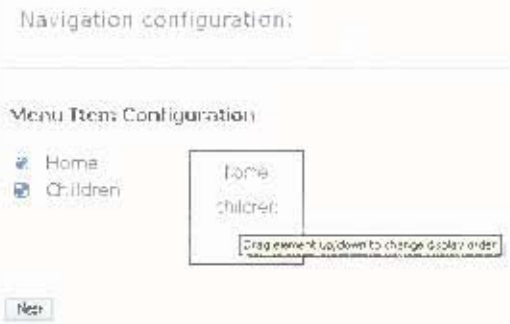

Figure 6.6: Navigation configuration

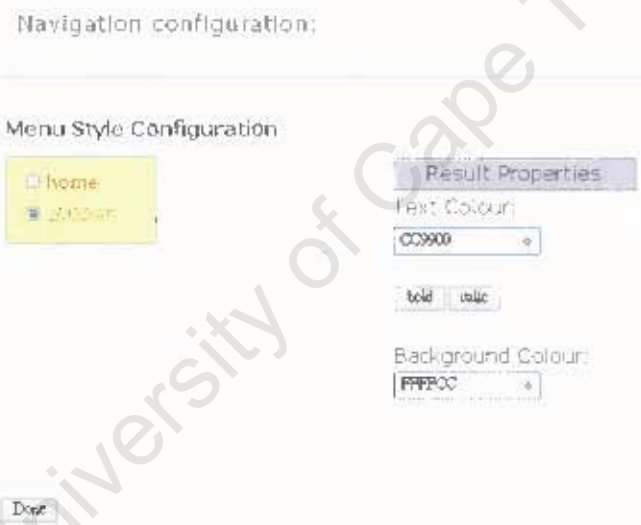

Figure 6.7: Navigation style configuration

links to other pages in the site structure; thus navigation is possible within the site created. The idea here is to define a general style for navigation menu for every page in the site; the only part of the navigation menu that differs are the actual links.

Two types of general link were identified: *Home* and *Children*. The home link takes a user straight back to the root page of the site, while Children links link to a page's direct child pages. Figure 6.6 shows how this is done in the system. The user is allowed to choose what type of links to have in the menu by selecting menu items (left); selected menu items will be dynamically rendered on the right. The order of display of menu items can also be changed by dragging these items in the rendered menu (right). The

system also provides some basic configuration of the menu style, such as text style, text colour and background colour. Figure 6.7 is a screenshot of the configuration page.

6.3 Interface design and rendering

In phase 2, interface design pages were designed to do only single page configuration (for Web services pages), but we are allowing the user to create a site structure in this phase, thus a single page interface design will no longer be suitable because it is impossible to pre-build interface design pages. Interface design pages need to be converted to a generic one so that any arbitrary type of page can be dealt with.

Figure 6.8: Normal page interface design

There are two main types of interface, namely normal and Web services; the main difference is that the Web service interface has a main page and a results page, while a normal interface page has only one single page. Thus the interface design page must be able to differentiate between the two types. This is achieved by passing different parameters to the interface design page when making the request, so the interface design page can display different options upon different types of page. Figure 6.8 and 6.9 illustrate how interface design pages renders two different options. Note the

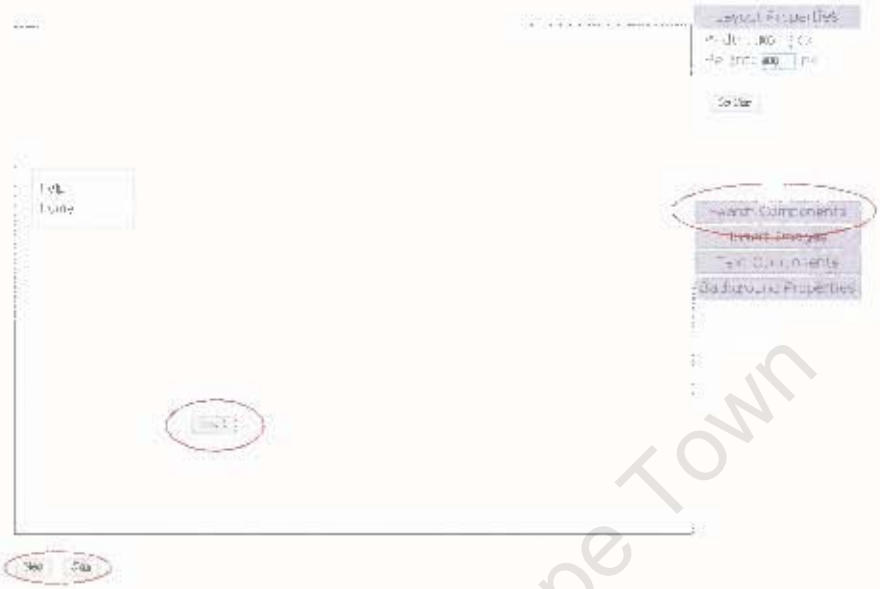

Figure 6.9: Web services page interface design

encircled elements in Figure 6.9; these options are specific to the Web service interface. In both figures, the navigation menu inside the layout area is the actual rendering of the menu defined by the user, so user can drag and place it to a suitable location in the page.

Two parameters are passed to the interface design page when making a request: *page name* and *level ID*. Page name is used to differentiate what type of interface options to display and level ID is used to generate a unique ID when saving the settings. Thus when rendering the actual pages, by supplying these two parameters, it is possible to retrieve the settings from the configuration XML file and re-create the pages. Illustration 6.2 is an example of the settings saved to the configuration XML file; note that the tag names are generated by the concatenation of page name and level ID.

6.4 Dual page mode for Web services page

As mentioned in the development cycle chapter, the Google style of results display is implemented in this phase. Two pages are dedicated to Web services: a main page contains search/browse boxes and a results page having same number of search/browse boxes as in main page, as well as a results display area. Thus when the interface design

```
<page>
    <home-0>
        ...
    </home-0>

    <search-1.1>
        ...
    </search-1.1>

    <searchresult-1.1>
        ...
    </searchresult-1.1>

    ...

</page>
```

Illustration 6.2: XML structure for page settings

page for a results page is requested, settings from the configuration XML file are retrieved in order to render whatever Web services components were defined. Users can then drag these components to a suitable location in the page. Figure 6.10 is what will be displayed if the button *Next* is clicked in Figure 6.9. The button *Skip* in Figure 6.9 is to skip the interface design process and jump straight to this design page, provided that interface design for the main Web services page was previously completed.

The main difference between this results interface design page and the main interface design page is the ability to define Web service result styles and their placement in the page. In phase 1 and 2 we have discussed Web service configuration where result display settings were configured and saved to a configuration XML file. These settings are retrieved here and used to render this Web service results block as seen in Figure 6.10.

The challenge of allowing a user to define styles for Web services arises when rendering the actual results page, with respect to applying these styles. Since PHP scripts were used to dynamically generate JavaScript in previous phases, it is possible for user PHP scripts to dynamically generate CSS scripts. With a little amendment to the generation of an XSL file, each record item is assigned with a unique ID. Thus the generated CSS scripts can be applied to corresponding record items.

Figure 6.10: Web services results page interface design

6.5 Status block

The status blocks were the last improvement made to the system. This block helps the user to monitor the current status of the configuration. Figure 6.11 illustrates how the status block looks before any configuration is carried out. Each small part of the status block represents a stage of configuration: an empty block means that the functionality is not yet configured, while a green tick means that the functionality is configured.

Figure 6.11: System status: Initial stage

The interface block has three statuses: empty block, yellow tick and green tick. Empty block means no interface is designed, yellow tick means only some pages are designed and green tick means all pages are designed. See Figure 6.12 for an example. Each of the blocks can also be clicked for fast accessibility. The *Preview* button is to preview the designed pages by popping up the root page in a new browser window: this button is disabled by default; it is only enabled when both *workflow* and *navigation* are

configured, and at least the root page is designed. See Figure 6.13 for example.

Figure 6.12: System status: Incomplete configuration on interface design

Figure 6.13: System status: Completed

6.6 Phase conclusion

6.6.1 What was found in this phase

From the last evaluation, it was found that the process of creating the site structure is not as intuitive as expected. During the evaluation, it was observed that participants could not make a mental connection from the site map rendered to the site structure they are creating. A possible solution could be to improve the rendering of the site map, but as was discussed in previous sections of this chapter, rendering of a site map is limited by the capability of Web browsers.

Another inconvenience found from observing the evaluation was the aligning of elements when designing interfaces. It was difficult for users to align elements either vertically or horizontally. One possible solution is to implement a grid system and have elements snap-to-grid. One participant also suggested a tool which can move elements according to a fixed axis, which can make element alignment easier.

6.6.2 Implication for future development

The results from the evaluation, in general, concentrated on improving the system's usability and functionality, but no major shortcomings were found in the system itself regarding its capability in creating a customised interface.

The system itself is only a prototype and needs a lot of improvements, but it is safe to say that this type of application is possible within a Web environment (AJAX application). The combination of server-side scripts and client-side scripts makes the development very flexible and many applications impossible in the past are made possible now with the current Web technologies.

Chapter 7

Discussion and Conclusions

After three rounds of implementation cycles, we have built a system that enables users to build a customised interface for digital libraries. During the development process, HCI and software engineering methodologies were applied, as well as some new Web technologies. We believe this type of application is particularly important for two reasons: the feasibility of developing a complex interface manipulation application in a Web browser (using AJAX) and the study of customisable user interfaces for digital libraries.

With the maturity of Web technologies, many previously impossible applications are now made possible in a Web browser. This includes dynamic content manipulation, content style manipulation and drag-and-drop mouse actions. Since AJAX applications are essentially Web applications, universal accessibility of the system through the Internet is another advantage for the type of application. As mentioned in previous chapters, browser compatibility remains a big problem for AJAX applications, because different browser vendors implement different subset of the standard and functions. Web developers will have to be considerate about this issue when developing this type of application.

From this research, it was found that Gecko based Web browsers, such as Firefox and Mozilla, provide very good support for W3CDOM, through their API; details of which can also be found easily online. Thus coding in a Gecko based Web browser (Firefox) was more straightforward than in, say, Internet Explorer. Internet Explorer support for DOM is poor and function names are different to the W3CDOM recommendation;

furthermore, documentation on the API for it is also hard to find. A lot more effort will be required if compatibility is to be maintained for both Gecko based and IE based Web browsers. If compatibility is not an issue, Gecko based Web browser would be the preferred choice for a development platform.

User interface customisation is still a difficult part in using digital libraries, especially for administrators who have no programming background. Thus this research demonstrated the feasibility of applying AJAX technology to interface builders and how it can be beneficial to the domain of digital libraries. Administrators are given the option to design a set of interfaces for their needs without having to learn complex skills.

7.1 Use of HCI during development

Developing the right system for a user is always difficult. In the case of this research,we employed HCI techniques to ensure as successful a result as possible. The application of UCD techniques called participatory design in phase 1 helped to focus the research to the important issue – the page design functionality. Through meeting with users and discussion on requirements and expectations for an interface system, important aims and issues were discovered at the outset.

Expert evaluation was then used to evaluate prototypes in phase 2 and 3. From the view of the experts in digital libraries, they can identify potential usability problems and offer recommendations for improvements. Thus a precise indication was obtained during the expert evaluation in phase 2 of this research. At the end of phase 3, expert evaluation was conducted again with the same group of experts – the purpose of this is to ensure that the system is up to the expectations of the experts.

The use of HCI techniques is a great help for developing a user oriented system, such as the one outlined in this research. It greatly shortens the time needed to identify user requirements and getting indications for improvements.

7.2 Use of current Web technologies

The major challenge for the development of the client-side of the system lies in the cooperation between JavaScript and DOM. JavaScript is used to obtain access to the

DOM object of a document – changes are then made to DOM object itself directly using (DOM) functions provided by the browser. The biggest problem encountered was the debugging of source code, because there is no compiler to pre-compile the code written, thus it is rather difficult to trace for bugs.

Two problems were encountered during the development, but no solutions are yet found. First was the rendering of the site map. The ideal way of rendering a site map is to display the site as a graph. However, due to the textual rendering characteristic of Web browsers, it is difficult to create a generic function to place elements in a spatial graph structure.

The other problem is the inter-communication between server and client at script level. This is needed when rendering the actual pages designed by user. Since it is possible that the resolution of the browser is higher than the pages designed, which is default to 800X600, the initial idea was to render the pages at the center of the browser. To do this, the browser resolution must be known by the server-side in order to compute the offset from left margin of the browser to the left page margin. Since we are rendering page elements by absolute positioning, this offset is needed to place the page in the center. The problem here is that the server has no access to the document DOM, which contains browser information and is accessible by client. Thus, if the client could pass the browser width to the server before the server renders the page, the offset can be computed and pages can be correctly placed in the center. To do so, the client needs to send this information to the server before requesting for a page using AJAX; however, this was not done in the project and is left for future work.

If the system is to be redesigned from scratch, the server-side scripting language of choice will not be PHP but JSP, because Java supports object-orientation and modularisation in a better manner. From experience, the code produced by using PHP can become very messy once the scale of the project becomes huge; it is also more difficult to maintain. XML support in JSP is also much better and mature when compared to PHP. If JSP is used, the server environment also needs to be changed from Apache to either J2EE server or Tomcat server. The trade off in switching scripting language will, of course, have to be further evaluated and assessed.

7.3 Future Work

At the end of the development phase of this research, it was found that the system built in this research is heading towards the direction of a Web page editor, such as Dreamweaver, but needing to run in a Web browser. It would be convenient if some of the features from traditional Web page editors, such as grid aligning, can be added to the system. There are also many features that were to be implemented in the system, but were then left out due to time constraints such as loading/saving of templates, reloading of designed pages and text styling.

The system currently works only with ODL components, but the use of the system in other application domains will be possible if an appropriate *Conveyor Module* is implemented.

In this research, only post-graduate students, who are specialised in digital library development, were invited for the rounds of evaluation. It would be best if the system can be further tested in a real-life environment by administrators; in this setup, more real-life behaviours of the system can be revealed and practical suggestions can be made to improve the system. However, we contend that our work provides a good foundation from which others can build and widen the scope of interface building for digital libraries.

Bibliography

[1] D. Bainbridge, J. Thompson, and I.H. Witten. Assembling and enriching digital library collections. *Proceedings of the 3rd ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 323–334, 2003.

[2] Waikato University Computer Science Department. Greenstone digital library software. Online. http://www.greenstone.org. Last visited: 1 June 2006.

[3] Apple Developer Connection. Developing dashboard widgets. Online, June 2005. http://developer.apple.com/macosx/dashboard.html. Last visited: 1 June 2006.

[4] H. Van de Sompel and C. Lagoze. Implementation guidelines for the open archives initiative protocol for metadata harvesting. Online, June 2002. http://www.openarchives.org/OAI/2.0/guidelines-static-repository.htm. Last visited: 1 June 2006.

[5] Information & Design. What is a participatory design workshop? Online, May 2003. http://www.infodesign.com.au/usabilityresources/ design/participatorydesign.asp. Last visited: 1 June 2006.

[6] K. Don, G. Buchanan, and I.H. Witten. *Greenstone3: A modular digital library.* Computer Science Department, Waikato University, November 2005. http://www.sadl.uleth.ca/greenstone3/manual.pdf. Last visited: 1 June 2006.

[7] L. Eyambe and H. Suleman. A component assembly approach to digital library systems. *Proceedings of the 2004 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries*, 75:15–22, 2004.

[8] DSpace Federation. Dspace digital repository system. Online. http://www.dspace.org. Last visited: 1 June 2006.

[9] C. Floyd. *A Systematic Look at Prototyping.* Springer Verlag, 1984.

[10] J.J. Garrett. Ajax: A new approach to web applications. Online, February 2005. http://www.adaptivepath.com/publications/essays/archives/000385.php. Last visited: 1 June 2006.

[11] H.M. Gladney, E.A. Fox, Z. Ahmed, R. Ashany, N.J. Belkin, M. Lesk, R. Tong, and M. Zemankova. Digital library: Gross structure and requirements (report from a march 1994 workshop). In *Proceedings of the First Annual Conference on the Theory and Practice of Digital Libraries.* IBM Almaden Research Center, Virginia Tech Dept. of Computer Science, June 1994.

[12] Google. Google products. Online. http://www.google.com/intl/en/options/. Last visited: 1 June 2006.

[13] A.S. Hornby. *Oxford Advanced Learners's Dictionary of current English.* Oxford University Press, fifth edition, 1995.

[14] J. Hughes, V. King, T. Rodden, and H. Andersen. Moving out from the control room: Ethnography in system design. In *Computer Supported Cooperative Work*, pages 429–439, 1994.

[15] Digital Library Research Laboratory. Open digital libraries components. Online. http://oai.dlib.vt.edu/odl/. Last visited: 1 June 2006.

[16] S. Mangano. *XSLT Cookbook.* O'Reilly, second edition, 2005.

[17] G. Marsden and D. Patel. Customizing digital libraries for small screen devices. *Proceedings of the 2004 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries*, 75:234–238, 2004.

[18] M.J. Muller. Participatory design: The third space in hci. *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, pages 1051–1068, 2002.

[19] National Information Standards Organization. *Dublin Core Metadata Element Set.* NISO Press, September 2001.

[20] Adobe Solutions Network. *Acrobat JavaScript Scripting Guide*, chapter 1, page 20. Adobe Systems Incorporated, September 2005.

[21] D.M. Nichols, K. Thomson, and S.A. Yeates. Usability and open-source software development. *Proceedings of the Symposium on Computer Human Interaction*, pages 49–54, 2001.

[22] J. Rintala. Document object model. Research Seminar Tik-111.590, Digital Media, November 2000.

[23] A. Rose, B. Shneiderman, and C. Plaisant. An applied ethnographic method for redesigning user interfaces. *Proceedings of the Conference on Designing Interactive Systems: Processes, Practices, Methods, & Techniques*, pages 115–122, 1995.

[24] K.M. Savetz. Getting the jump on javascript. *Mactech*, 12(7). http://www.mactech.com/articles/mactech/Vol.12/12.07/JavascriptIntro/. Last visited: 1 June 2006.

[25] R. Schmelzer. The pros and cons of xml. *Zapthink foundation report*, September 2001.

[26] Y. Shiran and T. Shiran. The document object model. Online, June 1999. http://www.webreference.com/js/column41/analyzecomplex.html. Last visited: 1 June 2006.

[27] B. Shneiderman. Direct manipulation: a step beyond programming languages. *IEEE Computer*, 16(8):57–69, August 1983.

[28] I. Sommerville. *Software Engineering.* Pearson Education Limited, 6 edition, 2001.

[29] H. Suleman and E.A. Fox. A framework for building open digital libraries. *D-Lib Magazine*, 7(12), December 2001. http://dlib.anu.edu.au/dlib/december01/suleman/12suleman.html.

[30] Hussein Suleman. Hussein's picture album. Online. http://www.husseinsspace.com/pictures.htm. Last visited: 1 June 2006.

[31] Hussein Suleman. *Open Digial Libraries.* PhD thesis, Virginia Polytechnic Institute and State University, November 2002.

[32] I.E. Sutherland. *Sketchpad: A Man-Machine Graphical Communication System.* Garland Publishers, New York, 1980.

[33] Tim Taylor. Drag & drop sortable lists with javascript and css. Online. http://tool-man.org/examples/sorting.html. Last visited: 1 June 2006.

[34] W3C. Document object model. Online, January 2005. http://www.w3.org/DOM. Last visited: 1 June 2006.

[35] Wikipedia. Xsl transformation. Online. http://en.wikipedia.org/wiki/XSLT. Last visited: 1 June 2006.

[36] F. Winberg. Usor - a collection of user oriented methods. Online, May 2003. http://www.nada.kth.se/cid/usor. Last visited: 1 June 2006.

[37] I.H. Witten, D. Bainbridge, and S.J. Boddie. Greenstone: Open-source digital library software. *D-Lib Magazine*, 7(10), 2001.

[38] N.C. Zakas. *Professional JavaScript for Web Developers.* Wrox, 2005.

[39] G. Zhang. Component-based software engineering. Technical Report 94-706-264, Information Institue, University of Zurich, August 2000.

[40] W. Zorn. Drag'ndrop & dhtml library. Online. http://www.walterzorn.com. Last visited: 1 June 2006.

Appendix A

Example

Here we will show, step by step, how to design interfaces with the final prototype of the system.

First of all, two XML files must be edited, *system.xml* and *odl.xml*, both files can be found in the *lib* folder. In system.xml, the path attribute of location tag under upload tag must correspond to the directory path on the server pointing to the upload directory in the main directory; for example, */public_html/carl/upload*. In odl.xml, the BaseURL of Web services must point to a valid Web services URL.

Now, we can start creating interfaces. The system is best viewed if screen resolution is 1280x1024 or above, since the pages we are designing are 800x600 by default. If we are to create a set of interfaces consisting of three pages, Home page, Search page and a Help page, the following steps illustrate the process:

1. **Define workflow**
   Here we define the site structure by adding desired pages. As illustrated in section 6.1, we create a site structure of three pages. Figure A.1 shows the screenshot.

2. **Configure Web service**
   The next step is to configure Web services; Figure A.2 shows the relevant screenshot. Select the Dublin Core items on the left by clicking on the check boxes, then drag the blocks on the right to decide the display order. The paging position and the number of entries to be displayed per page can also be changed.

3. **Configure navigation menu**
   The navigation menu will be configured in this step. This consists of two sub steps: the first is to define what links are to be displayed and the second is the style of the menu. In Figure A.3, you can select the type of link to be displayed by clicking on the checkboxes. The elements in the block on the right can be dragged to determine the display order. Click on *Next* once you are done with the setting.

   Here in Figure A.4, you will be able to change the background colour of the menu. The style of the links can be done by selecting a link type and then choosing a colour from the dropdown box or checking bold/italic options.

4. **Build interfaces**
   When the *Build Interface* link in the side menu is clicked, the site structure of the interface pages will be shown, such as in Figure A.5. Each of the elements in the site structure can be clicked, which will take you to its design page.

   Figure A.6 shows the design page of the *Home page* when it is clicked. As explained in section 5.2, the big block in the center is the layout area where you will be designing the page. On the right is the toolbar, where each of the tabs can be clicked to expand or collapse. The image tab allows you to insert an image by clicking on the thumbnails; text tab allows you to insert single texts or paragraphs; background tab lets you define background colour or image; there is also a remove button in the background tab which removes the background image. Each of the elements in the layout area are drag-able and can be deleted by double clicking on the element.

   Figure A.7 is the design page for the search Web services. There is an extra tab in the toolbar which lets you insert Web services fields. Note that there are two buttons below the layout area, *Next* and *Skip*. When you are done with designing the page for the first time, you should click on Next to proceed to the design page for Web services results page, as shown in Figure A.8. The purpose of the Skip button is to let you skip to the design page of Web services results page, if the Web services page was already designed.

Figure A.1: Define workflow

5. **Test interfaces**

   Once the Home page is designed, the *Preview* button in the status block will be enabled and the interface block be given a yellow tick. The interface block will be given a green tick when all pages are designed, as shown in Figure A.9. When the preview button is clicked, a new browser window will be opened with Home page loaded.

Figure A.2: Configure Web services

Figure A.3: Configure navigation menu link

Figure A.4: Configure navigation menu style

Figure A.5: Interface design: site map

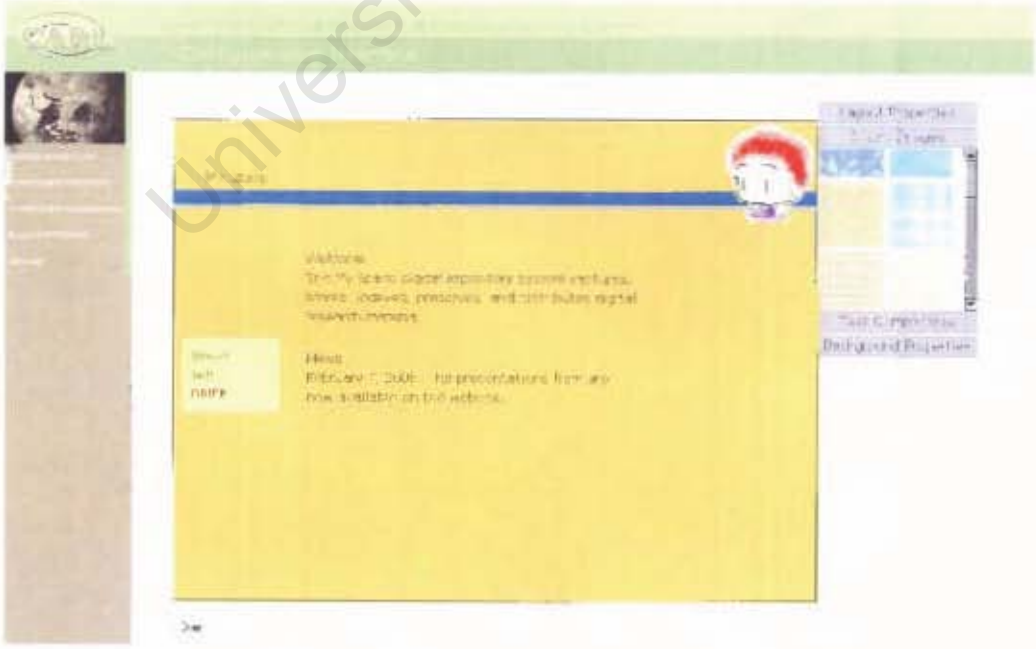

Figure A.6: Interface design: home page

Figure A.7: Interface design: search page

Figure A.8: Interface design: search results page

Figure A.9: Status block: completed