UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Design implications of an online collaborative workspace developed using open source software

By

Paul Bothma

Dissertation

Submitted as partial fulfilment of the requirements for the degree

MIS Multimedia

In the

Department of Information Science

Faculty of Engineering, Built Environment and Information Technology

University of Pretoria

Study leader: Prof. Johannes Cronje

November 2006

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Summary

Design implications of an online collaborative workspace developed using open source software

This thesis reports on a pilot project which was developed to meet the needs of specific research groups for a collaborative workspace. The main components of the project include a digital library, online discussion forum, adaptive hypermedia engine and a statement database.

Such a system was deemed necessary because of today's global network of researchers who are divided by both location and time. The need therefore exists for a system which allows researchers to work on a project without the need to be in the same physical location. Such a system is called an online collaborative workspace. An online collaborative workspace allows researchers to work together on projects by providing various means of sharing information and resources.

The main research question addressed in this study is:

- What are the user requirements and design specifications of an online collaborative workspace developed in open source software?

In order to answer this main question, the following sub-questions need to be addressed:

- What are the main design principles and functionalities of such a collaborative workspace?
- How can they be integrated in developing a modular open source framework?
- To what extent can such a framework be customised for implementation in new or existing collaborative projects?
- What are the usability requirements of such an online collaborative workspace and how should one go about to test the usability of such a framework?

Before being able to accurately determine the user requirements of an online collaborative workspace, we first need to determine which features exist in current systems. The main question driving the literature survey therefore is:

- What are the main components of current online collaborative workspaces as reported in the literature?

Chapter 2 of this report is devoted to identifying and researching the various

components that define an online collaborative workspace. Each of these components are investigated in detail by means of informal interviews as well as a literature survey. The components that were identified include a digital library, online discussion forum and collaborative workspace. In addition to identifying the various components, this chapter also addresses the motivation for the use of open source software and the importance of usability.

With the information obtained in chapter 2, chapter 3 sets out to develop a framework which addresses each of the components that were identified. The various features and properties of each component were identified and decisions were made with regard to the best method of implementation. In addition to the components that were developed, this chapter also focuses on various design implications such as modularity and the use of open source software.

Chapter 4 describes the various projects where the framework was implemented. For each of the projects the user requirements, customisations and usability issues are addressed. This chapter highlights the framework's ability to be customised to suit the diverse needs of research projects.

The final chapter provides a set of conclusions summarising how the main objectives of the study were addressed. Moreover, the limitations of the study are addressed and an outline is provided as to how these limitations are to be addressed in future work.

Table of Contents

1 Chapter 1 1
1.1 Research questions..1
1.2 Research methodology...2
1.2.1 Informal survey...2
1.2.2 Literature survey...3
1.2.3 Project development..4
1.3 Limitations...5
1.4 Chapter division..5
1.4.1 Chapter 1..5
1.4.2 Chapter 2..6
1.4.3 Chapter 3..6
1.4.4 Chapter 4..6
1.4.5 Chapter 5..6
1.5 Summary..6
2 Chapter 2 – Literature survey...8
2.1 Introduction..8
2.1.1 Digital library...8
2.1.1.1 Brief overview..8
2.1.1.2 Characteristics of a Digital Library..9
2.1.1.3 Advantages of a Digital Library...11
2.1.1.4 Open Archives Initiative..12
2.1.1.5 Current implementations...13
2.1.1.5.1 EPrints...13
2.1.1.5.2 Greenstone...14
2.1.1.5.3 DSpace..16
2.1.1.6 Summary findings...18
2.1.2 Adaptive hypermedia...19
2.1.2.1 Brief overview...19
2.1.2.2 User profiling/models..20
2.1.2.3 What can be adapted...21
2.1.2.3.1 Link level adaptation..21
2.1.2.3.2 Content level adaptation...22
2.1.2.4 Recommendation system...23
2.1.2.5 Adaptive hypermedia methodologies......................................24
2.1.2.5.1 Dexter model...24
2.1.2.5.2 AHAM...25
2.1.2.6 LinkBases and link services..26
2.1.2.7 Current implementations..27
2.1.2.7.1 The GUIDE system...27
2.1.2.7.2 ELM-ART...29
2.1.2.8 Summary findings..30
2.1.3 Collaborative Workspaces...31
2.1.3.1 Introduction...31
2.1.3.2 Characteristics of collaborative workspaces..............................31
2.1.3.2.1 Session vs. document centric design.....................................31
2.1.3.2.2 Ad hoc vs. formal systems..32
2.1.3.2.3 Features of collaborative workspaces....................................33
2.1.3.3 Current implementations..34

2.1.3.3.1 phpGroupWare....34
2.1.3.4 Summary findings....35
2.2 Open Source Software....36
2.2.1 A brief history of free/open source software....36
2.2.2 The open source model....38
2.2.3 Open source development tools....39
2.2.4 Open source projects....40
2.2.4.1 The Apache Software Foundation....40
2.2.4.2 Linux....41
2.2.4.3 The Mozilla Foundation....42
2.2.5 Motivation for the use of open source software....43
2.3 Usability....45
2.4 Literature survey findings....46
2.4.1 Critical reflection on the literature....46
2.4.2 Open Source Software....49
2.5 Summary....50
3 Chapter 3 - Framework architecture and research methods....51
3.1 Introduction....51
3.2 Design principles....53
3.2.1 Modularity....53
3.2.2 Three-tiered architecture....53
3.2.3 Implemented technologies and software....54
3.3 Functional components....55
3.3.1 Content module....55
3.3.1.1 Overview....55
3.3.1.2 Classes....56
3.3.1.2.1 Module....56
3.3.1.2.2 Loader....56
3.3.1.2.3 Metadata....57
3.3.1.2.4 Link....57
3.3.1.2.5 Section....58
3.3.1.2.6 Page....59
3.3.1.2.7 Subject....60
3.3.1.3 Clustering....60
3.3.1.4 Integration of modules....61
3.3.1.4.1 Modules and components....62
3.3.1.4.2 Interface....62
3.3.1.4.3 Java Server Pages....63
3.3.1.5 Database....63
3.3.1.6 Adaptation....65
3.3.1.6.1 Adaptive presentation....65
3.3.1.6.2 Adaptive navigation....66
3.3.1.7 Recommendation system....66
3.3.1.7.1 Requirements and history....66
3.3.1.7.2 Personal preferences....67
3.3.1.8 Content Management....68
3.3.1.8.1 Content submission....68
3.3.1.8.2 Theme management....68
3.3.1.8.3 Link management....69
3.3.1.8.4 Page requirements....69
3.3.1.9 Configuration....70

3.3.1.9.1 Standard settings....70
3.3.1.9.2 Additional settings....70
3.3.2 Discussion forum....71
3.3.2.1 Overview....71
3.3.2.2 Classes....71
3.3.2.2.1 Module....71
3.3.2.2.2 Loader....72
3.3.2.2.3 DiscussionMessage....72
3.3.2.2.4 DiscussionTopic....73
3.3.2.2.5 Discussion....73
3.3.2.3 Integration....73
3.3.2.4 Database....74
3.3.2.5 XML & XSL....74
3.3.2.6 Functionalities....74
3.3.2.7 Forum management....75
3.3.2.8 Configuration....75
3.3.2.8.1 Standard settings....76
3.3.2.8.2 Additional settings....76
3.3.3 Repository and workspace....77
3.3.3.1 Overview....77
3.3.3.2 Classes....77
3.3.3.2.1 Module....77
3.3.3.2.2 Loader....78
3.3.3.2.3 MetadataField....78
3.3.3.2.4 Metadata....80
3.3.3.2.5 File....80
3.3.3.2.6 Hierarchy Explorer....80
3.3.3.2.7 Directory....81
3.3.3.2.8 DocumentSearch....81
3.3.3.2.9 Document....82
3.3.3.2.10 RepositoryInstance....83
3.3.3.2.11 Open Archives Initiative (OAI)....83
3.3.3.3 Integration....83
3.3.3.4 Hierarchy & Document storage....83
3.3.3.5 Presentation & Functionalities....84
3.3.3.5.1 Metadata....84
3.3.3.5.2 Document content....85
3.3.3.5.3 Available functionalities....85
3.3.3.6 Search....86
3.3.3.6.1 Search interfaces....86
3.3.3.6.1.1 Simple....86
3.3.3.6.1.2 Advanced....87
3.3.3.6.1.3 Full text....87
3.3.3.7 Database....87
3.3.3.8 Document management....89
3.3.3.8.1 Document submission....89
3.3.3.8.2 Editing, revision, accepting and removal of documents....89
3.3.3.8.3 Directory management....90
3.3.3.9 Configuration....91
3.3.3.9.1 Standard settings....91
3.3.3.9.2 Additional settings....91

3.3.4 Statement database....93
3.3.4.1 Overview....93
3.3.4.2 Classes....93
3.3.4.2.1 Module....93
3.3.4.2.2 Loader....93
3.3.4.2.3 Link....93
3.3.4.2.4 Metadata....94
3.3.4.2.5 Statement....94
3.3.4.2.6 StatementDatabase....95
3.3.4.3 Integration....95
3.3.4.4 Presentation & Functionalities....95
3.3.4.4.1 Browsing databases....95
3.3.4.4.2 Viewing statements....96
3.3.4.5 Database....96
3.3.4.6 Statement database management....96
3.3.4.7 Configuration....97
3.3.4.7.1 Standard settings....97
3.3.4.7.2 Additional settings....98
3.4 Supporting core modules....98
3.4.1 Graphical User Interface component....99
3.4.1.1 Overview....99
3.4.1.2 Classes....99
3.4.1.2.1 Loader....99
3.4.1.2.2 FieldValidator....99
3.4.1.2.3 Link....100
3.4.1.2.4 Expression....101
3.4.1.2.5 Interface....101
3.4.1.2.6 InterfaceComponent....102
3.4.1.3 XML & XSL....102
3.4.1.4 JSP template....103
3.4.1.5 Configuration....104
3.4.1.5.1 Standard settings....104
3.4.1.5.2 Additional settings....105
3.4.2 User component....105
3.4.2.1 Overview....105
3.4.2.2 Classes....106
3.4.2.2.1 Loader....106
3.4.2.2.2 ProfileEntry....106
3.4.2.2.3 Profile....107
3.4.2.2.4 User....108
3.4.2.3 Functionalities....108
3.4.2.3.1 My homepage....109
3.4.2.3.2 View profile....109
3.4.2.3.3 View recommendations....110
3.4.2.3.4 Create a new project....110
3.4.2.3.5 Manage users....110
3.4.2.4 User management....110
3.4.2.5 Database....111
3.4.2.6 Configuration....111
3.4.2.6.1 Standard settings....111
3.4.2.6.2 Additional settings....112

3.4.3 Plugins....112
3.4.3.1 Overview....112
3.4.3.2 Plugin chain....113
3.4.3.3 Examples....114
3.4.3.3.1 AlterMessage....114
3.4.3.3.2 NotifyAdministrator....114
3.4.3.3.3 AlterContent....114
3.4.3.3.4 NotifyOriginalSender....114
3.4.4 Security....115
3.4.4.1 Overview....115
3.4.4.2 Security model....115
3.4.4.2.1 Role-based security model....115
3.4.4.2.1.1 Default....115
3.4.4.2.1.2 User....115
3.4.4.2.1.3 Worker....116
3.4.4.2.1.4 Admin....116
3.4.4.2.1.5 Super....116
3.4.4.2.2 Authorization....116
3.4.4.2.3 Authentication....116
3.4.4.3 Navigation....117
3.5 Research methodology....117
3.5.1 Research approach....117
3.5.1.1 What is development research and why do we use it?....118
3.5.1.2 What are the most common attributes of development research?....118
3.5.1.3 Why can this project be seen as development research?....119
3.5.2 Research method....119
3.5.2.1 Sampling....120
3.5.2.2 Data collection....120
3.5.2.3 Interpretation....121
3.5.2.4 Limitations....122
3.6 Summary....123
4 Chapter 4 - Implementations....124
4.1 Introduction....124
4.2 Implementations....125
4.2.1 DISSAnet....126
4.2.1.1 Overview....126
4.2.1.2 Client requirements....127
4.2.1.3 Features and functionalities....127
4.2.1.4 Lessons learnt....129
4.2.2 IKS....130
4.2.2.1 Overview....130
4.2.2.2 Client requirements....131
4.2.2.3 Implementation and customisations....131
4.2.2.4 Lessons learnt....133
4.2.3 IFLA-KM....135
4.2.3.1 Overview....135
4.2.3.2 Client requirements....136
4.2.3.3 Implementation and customisations....136
4.2.3.4 Lessons learnt....136
4.2.4 The Wellness Firm....138
4.2.4.1 Overview....138
4.2.4.2 Client requirements....139

4.2.4.3 Implementation and customisations....139
4.2.4.4 Lessons learnt....140
4.3 Summary....141
5 Chapter 5 – Conclusions and recommendations....143
5.1 Introduction....143
5.2 Summary findings regarding research objectives....143
5.2.1 What are the main components of current online collaborative workspaces as reported in the literature?....143
5.2.2 What are the main design principles and functionalities of such a collaborative workspace and how can they be integrated in developing a modular open source framework?....145
5.2.3 To what extent can such a framework be customised for implementation in new or existing collaborative projects?....146
5.2.4 What are the usability requirements of such an online collaborative workspace and how should one go about to test the usability of such a framework?....146
5.2.5 What are the user requirements and design specifications of an online collaborative workspace developed in open source software? 147
5.2.5.1 User requirements....147
5.2.5.2 Design specifications....149
5.2.5.3 Open source software....151
5.3 Reflections....152
5.3.1 Methodological reflection....153
5.3.2 Substantive reflection....153
5.3.3 Scientific reflection....154
5.4 Recommendations....155
5.4.1 Policy and practice....155
5.4.2 Future work....157
5.5 Final conclusions....158
Bibliography 163
Appendix 168

1 Chapter 1

This thesis reports on a pilot project which was developed to meet the needs of specific research groups for a collaborative workspace. The main components of the project include a digital library, online discussion forum, adaptive hypermedia engine and a statement database.

Such a system was deemed necessary because of today's global network of researchers who are divided by both location and time. The need therefore exists for a system which allows researchers to work on a project without the need to be in the same physical location. Such a system is called an online collaborative workspace. An online collaborative workspace allows researchers to work together on projects by providing various means of sharing information and resources.

Although many projects exist that address specific requirements of the research community the literature survey and interviews performed during this study, indicated that the existing systems do not necessarily satisfy all the needs of users. Moreover, most systems are built using proprietary software, thus limiting customisation of the components to suit an institution's or a specific research group’s unique needs. Another result of proprietary software is the cost involved with development, therefore making the final system unobtainable to most research groups.

The main reason, therefore, for the development of this framework was to create a prototype, using open source software, that satisfies the needs of specific research communities by addressing the shortcomings of the existing systems.

1.1 ***Research questions***

As a result of these limitations, it became evident that there is a gap in the solutions available to today's researchers. The main research question addressed in this study therefore is:

- What are the user requirements and design specifications of an online collaborative workspace developed in open source software?

In order to answer this main question, the following sub-questions need to be addressed:

- What are the main design principles and functionalities of such a collaborative workspace?
- How can they be integrated in developing a modular open source framework?

- To what extent can such a framework be customised for implementation in new or existing collaborative projects?
- What are the usability requirements of such an online collaborative workspace and how should one go about to test the usability of such a framework?

Before being able to accurately determine the user requirements of an online collaborative workspace, we first need to determine which features exist in current systems. The main question driving the literature survey therefore is:

- What are the main components of current online collaborative workspaces as reported in the literature?

1.2 ***Research methodology***

Due to the fact that the final outcome of this research project was to develop a framework that meets specific user requirements, it was decided to follow a developmental approach. The research methods that were used, therefore, consisted of first identifying the user requirements of today's research community. This was done by combining the findings of both an informal survey and a literature review of existing systems. The literature survey consisted of consulting both scholarly research and existing systems. An informal survey provided invaluable insights as to exactly what features are required by today's research community. Lastly, the framework, meeting the user requirements, could therefore only be developed after gaining a better understanding of the research field.

1.2.1 Informal survey

The need for the development of a framework that supports online collaboration of researchers, arose out of the needs of DISSAnet and IKS research projects. These are two research projects which expressed the need for a system which would allow its members to share documents and thoughts amongst each other. DISSAnet's focus was on providing researchers a centralised repository where conference proceedings could be stored and retrieved. IKS, on the other hand, required a system where indigenous knowledge from across Southern Africa could be stored and made available to research community. The requirements for such a system were obtained by interviewing the people involved and also by performing a literature survey. The interview process consisted of talking to the researchers and discussing their needs as a research group and how one could go about addressing those needs.

1.2.2 Literature survey

Before embarking on a research project, it is important to first ascertain whether any similar systems exist, and if so, what functionalities they provide. This is made possible by performing a literature survey, which is described by Fouche [Fouche 2002] as a scrutiny of all relevant sources of information. A literature survey allows one to investigate other research projects' methodologies and practices, which assists one in determining the best course of action. Moreover, by performing a literature survey, researchers are made aware of other projects in the same area of study, hence duplication of research efforts can be avoided.

An online collaborative system is comprised of a collection of services which allow users to work collaboratively on a project. The services that were identified included digital libraries, adaptive hypermedia, online discussion forums and collaborative workspaces. Many such services already exist in the form of either free or proprietary systems. Examples of each such service were identified and researched to determine their features, specifications and success. The research process involved with each of the systems comprised of sourcing documentation, examples and research papers from the Internet, including digital libraries and online journals. Typical search terms that were used included “adaptive hypermedia”, “collaborative workspaces”, “digital library”, “linkbases”, “link services” and “ad hoc information systems”. This information was then used to provide amongst others an overview of each service as well as their features, advantages and characteristics. Lastly, examples of current implementations of each such system is provided, consisting of an overview of the implementation and its key features and functionalities. As stated, most of the research was done using sources obtained from the Internet. One such source is the Association of Computing and Machinery (ACM), which provides researchers as well as scholars an invaluable source for conference proceedings, journals and articles in their digital library. By reviewing the existing services needed to develop an online collaborative workspace, a better understanding was obtained as to how these services could be integrated to provide a framework capable of bringing project members together.

Another focus of the survey involved the research of Open Source Software (OSS), which played a key role in the development of the framework. A review of both old and new literature provided enough insights on the benefits on Open Source Software and also the strategies to be followed when building such a framework. Only after identifying the services needed to provide an online collaborative system could the decision be made as to what software would be needed in the implementation of such a system. Included in the list of required software are a relational database system, web service and various programming languages. Examples of each of these requirements were reviewed and a decision whether to adopt the specific technology was made based on the functionality, flexibility and scalability of the technology.

1.2.3 Project development

Development and implementation of the framework consisted of first deciding on a broad set of specifications, after which a prototype was developed. The prototype was reviewed by the group of researchers who assisted in the analyses of the user requirements of the online collaborative workspace. An iterative development and review process was followed to ensure that the final implementation of the framework would satisfy the needs of the specific research communities.

a) Broad specification

   A broad specification, comprising a list of features, was designed after reviewing existing systems and interviewing a group of researchers. I decided to adopt a broad specification in stead of a detailed specification, as this allowed for the rapid development of a prototype which could be enhanced with each development interval.

b) Prototyping

   The use of prototyping allowed me to rapidly develop a system that can be tested and reviewed by, for instance, a group of researchers with an interest in the project [Strydom 2002]. By adopting a method of continuously submitting a prototype system for review, I could quickly and effectively address any issues which might arise. A prototype system would typically start by implementing the list of broad specifications that were identified, after which each feature would be tailored to suit the needs of the users. By following this breadth-first approach, I could quickly grasp the impact and scale of the system.

c) Usability

   A collaborative workspace's purpose is to allow project members to interact with one another using a virtual medium. It is therefore important for that medium to assist the users in their task as effectively and efficiently as possible. An important consideration of any framework that is to be used by users is usability. Usability is defined by Nielsen [Nielsen 1993] as not only a single one dimensional concept, but rather a multi-dimensional collection of attributes. The accepted list of attributes of usability include learnability, efficiency, memorability, low error rate and satisfaction. When developing any system, it is important to understand the user's needs, abilities and the scenario in which such a system is to be used. An important step of determining the usability of a system is done through iterative user testing [Shneiderman 2005]. Nielsen [Nielsen 1993] proposes "discount usability" which holds that when the number of test users is limited , it is important to ensure that the participants are representative of the expected user group.

1.3 ***Limitations***

The main user requirements that were identified by the informal interviews were for an online collaborative workspace that provided researchers with a structured, asynchronous system to assist then in sharing information and ideas. As a result certain features such as a shared whiteboard and synchronous chat, which were identified in the literature survey, were not implemented in the framework. Certain features that were not implemented in the current framework were, however, identified to be of great use to researchers and will certainly be addressed in future work and research.

In addition to the functional limitations, there also existed limitations with regard to the usability testing of the framework. Because of the fact that a developmental approach was followed, the usability testing of the framework consisted of working closely with the experts involved with the study and was therefore tested throughout the various stages of the development process. Expert testing, instead of exhaustive user testing, was the preferred choice for usability testing due to the complexity and the constant changing of user requirements of the framework. Another motivation for expert testing is the fact that not all of the functionalities provided by the framework have been implemented in the various projects. One such an example is the adaptive hypermedia engine provided by the framework which has, at the time of writing, not yet been implemented in any of the research projects; it was, however, identified by the original DISSAnet research project as a requirement but was not utilised due to various reasons. The final chapter of this study addresses the limitations that were identified and how future work could address these limitations.

1.4 ***Chapter division***

1.4.1 Chapter 1

As an introduction to the subsequent chapters, the first chapter provides the main objective of this study as well as a set of sub-problems. The chapter continues with the proposed research methodology for this study in order to solve the various problems that were identified. Following the research methodology is a short discussion of the areas and topics that are not covered by the study. Ending off the chapter is an outline of the chapters comprising this research study.

1.4.2 Chapter 2

This chapter is devoted to gaining a better understanding of the components that define an online collaborative workspace. Each of the components was identified and researched by means of a literature survey. Digital libraries, online discussion forums, collaborative workspaces, adaptive hypermedia and open source software were identified as the main areas of interest. The literature survey is divided into a number of sections; each addressing a different field.

1.4.3 Chapter 3

With the information obtained from the literature survey, it was possible to define a set of specifications to be addressed by the framework to be developed. This chapter describes the various components of the framework and how they address the design specifications. The components involved include an adaptive hypermedia engine, a digital library, online discussion forum and statement database. Each component is described in detail and includes the full technical specification, uses and configuration.

1.4.4 Chapter 4

The completed framework was implemented in various research projects to test its ability to meet the users' functional as well as usability requirements. This chapter describes the various instances where the framework was implemented. Each implementation was customised to meet the specific research project's needs which were identified during an interview process.

1.4.5 Chapter 5

The final chapter provides a set of conclusions summarising how the main objectives of the study were addressed. Moreover, the limitations of the study are addressed and an outline is provided as to how these limitations are to be addressed in future work.

1.5 ***Summary***

This chapter first identified the importance and main research objectives of the research study, viz. determining the user requirements and design specifications of an online collaborative workspace using open source software. Following the introduction and research questions the research methodology outlined the processes which will be followed in order to answer the research questions that were identified, and included an informal interviews, literature survey and lastly project development. The informal interviews consisted of obtaining a list of user requirements from research experts. Following the interviews, the literature survey provided a more in-depth understanding of the various components that are needed in an online collaborative workspace as they exist today. Lastly the process of developing and testing such a prototype system consisted of working closely with the research experts and continuously revising the user requirements and evaluating the usability of the framework.

Included in this chapter is a list of limitations that were identified through the gap analysis between the user requirements as obtained from the informal interviews and the components that were identified during the literature review. Lastly, the chapter division provided an outline of what is to follow in the rest of the study.

The following chapter is devoted to identifying the various components that are needed to develop a successful online collaborative workspace. This is done by performing a literature survey in which conference proceedings, research publications and various existing systems are evaluated in order to obtain a clear understanding of what is being done in the field of online collaborative workspaces.

2 Chapter 2 – Literature survey

2.1 *Introduction*

In chapter 1 the question driving the literature survey was formulated as:

- What are the main components of current online collaborative workspaces as reported in the literature?

This chapter aims to answer this question by identifying and investigating the various components of online collaborative workspaces and open source technologies that can assist in the development of such a system.

Firstly, the focus is on the main components which form the base of such a system. The list of components comprises of a digital library, adaptive hypermedia and collaborative workspaces. A brief overview describing each component and also its characteristics is provided. A review of current implementations, in both scholarly research as well as commercial systems, of the various components provides insightful information into the features of these components and how they can be adapted to suit the needs of an open source collaborative workspace.

This chapter is concluded with the history and characteristics of the open source software movement as well as the open source model used to develop free software. The open source software movement has ensured that there exists a multitude of tools and applications which makes the development of an online collaborative workspace based on open source a reality. Components of a collaborative workspace

2.1.1 Digital library

2.1.1.1 Brief overview

A common misconception is that a digital library is merely a digital version of a real-world library. There are, however, numerous differences between digital and real-world libraries. The most obvious difference is of course the fact the digital libraries are digital and in most cases do not represent a physical library.

As a result, digital libraries do not necessarily adhere to real-world principles. For example, various new ways of storing and representing documents need to be devised to allow for the complex components of modern-day documents. Although many digital libraries try to imitate real-world libraries by using elements and principles well known to users, these are not always successful as computer screens and the interaction with computers have certain limitations [Wiederhold 1995].

As an example we investigate the use of a card based metaphor, such as the use of Dewey Decimal cards in libraries, as a means to display search results. Whilst the metaphor is known to anyone who has used a library before, the use of such visual elements can hinder the user's ability to navigate the results due to, for example, the difficulty in quickly and easily selecting search results. In addition to hindering the user's experience, such elements are also not printable thus limits the ability to file or use search results in any other format as that which it is originally presented in. As a result, developers of digital libraries have to move away from the paradigms of real-world libraries and create new ways of allowing users to navigate the information space of the digital library.

Wiederhold [Wiederhold 1998] identifies other social differences between digital and real-world libraries; however, these are beyond the scope of the research.

2.1.1.2 ***Characteristics of a Digital Library***

Levy [Levy 1995] identifies three core components of a library, viz.

- *Documents*
  These are the physical documents or artefacts that are contained within the library and can vary in their rate of change and duration/lifespan.
- *Technology*
  The artefacts in the library are created by humans using some form of technology, be it pen and paper or any other form of craft.
- *Work*
  Humans are needed to create the library and manage the collections of work. In addition to creating the library, humans are also the consumers of the information contained within the library. Both the creation and use of the library thus entails work being done by humans.

By comparing digital libraries with real-world libraries, one can see that in essence the three core components remain unchanged. There does, however, exist some differences in the implementation of each component. For example, the technologies used to create artefacts for the digital library will be digital and the underlying technologies supporting the library will also be digital. Thus, for a digital library the core components' attributes would become:

- *Documents*

A library can contain either a single fixed version of a document, or there could be multiple versions of a document that can change as time progresses.

- *Technology*
  The underlying technologies of a digital library and its artefacts will inherently be of a digital form.
- *Work*
  Most work done by humans on the digital library, be it managing or research, will be done by individuals working alone. For example, a student could sit in his/her room and use the digital library without any interaction with humans.

At the 1998 DLib Working Group on Digital Library Metrics [DLib 1998] the following properties for a digital library were proposed:

- *Provide a collection of services*
  A digital library must serve a broad range of users such as researchers, publishers, managers and even machines. It is therefore important for any digital library to expose a collection of services that can be utilised by these users to satisfy their specific needs. Examples of services include storing, retrieving, indexing and browsing of information.
- *Support to users in dealing with information objects*
  A digital library must support users in dealing with the collection of information objects by not only allowing users to access information but also to manage the vast collection of documents, users and hierarchies. In addition to allowing humans to use the digital library, a digital library must also be made accessible to automated queries that are made on behalf of users or for automated harvesters.
- *Provide a collection of information objects*
  The basis for a digital library is the information that it contains which forms the content for the library. In a traditional library information objects were limited to physical documents and articles, whereas digital libraries can contain electronic representations of traditional documents but also live data such as sensory information.
- *Organization and presentation of information objects*
  In real-world libraries users have to locate a book using the author, title, date and directions to the physical location of the bookshelf containing the book. Digital libraries work in much the same manner, requiring users to have some information pertaining to the desired document, such as a part of the document's

title or the author. However, instead of providing the physical location the book on a shelf, a digital library provides a link to the document within the repository. Ongoing research is required in order to ensure that digital libraries provide access to information in a manner that is most suitable for humans and that comply with usability guidelines.

- *Be available directly or indirectly*
  Information contained within a digital library can either be the full text of the document itself, or a reference to a document in another digital library, or simply the physical location of the book in a real-world library.

- *Available via electronic/digital means*
  Each entry in a digital library must at least be accompanied by metadata information that contains information relating to the document; however, the document's full text need not necessarily be stored in the digital library.

The list of properties that were identified provided a starting point for the digital library component of the online collaborative workspace. There exists a correlation between the before mentioned list of features and attributes and the user requirements that were identified during the informal surveys.

2.1.1.3 ***Advantages of a Digital Library***

A digital library can have certain advantages, if implemented correctly, above a traditional real-world library. Some advantages worth noting include:

- *Support for enterprise and work group activities* [Anderson 1997]
  Corporations are increasingly making use of digital libraries to store documents relating to their business practices and models in digital libraries to allow access to both employees and consumers.

- *Searching* [Lesk 1995], [Thong 2004]
  A major benefit of digital libraries is the ability to create full text indices of a document's text which allows users to locate documents based on not only limited metadata but also the content.

- *Accessibility* [Lesk 1995], [Thong 2004]
  A single copy of a document can be accessed simultaneously by numerous users across the globe at any given time, thus making the limitations of time and space obsolete. Also, the document can be delivered in a format that is most suitable to the users, for instance if a user accesses the digital library from a mobile device,

the document can be formatted appropriately.

- *Preservation* [Lesk 1995]
  Digital content can be copied without error and without damaging the original source. This reduces the need for storing original copies in a secure location, which greatly increases the cost incurred by real-world libraries.
- *Tracking* [Thong 2004]
  A digital library can provide detailed statistics and trends relating to which documents are accessed by certain user groups. This provides administrators with useful information that can help with the improvement of the digital library.
- *Associative linking and annotations* [Miles-board 2004]
  A digital library can be enhanced by allowing users to add annotations and associative links that can provide useful cross-linking and value added information to existing documents. Researchers can then use these associative links and annotations in the search for information.
- *Multimedia/Rich media* [Lesk 1995]
  Digital libraries are not bound by the same limitations that real-world libraries face when it comes to the format of the information objects. Real-world libraries are, typically, limited to printed media and a limited set of audio-visual aids. However, digital libraries can contain a vast array of information objects ranging from static documents to interactive media and possibly even games. The availability of these new rich media information objects can greatly assist users in their information gathering needs and research.

The efficiency and effectiveness of the digital library could be greatly improved by focussing on the various advantages that were identified during the literature survey. These advantages have to be kept in mind during the development stages to ensure that the benefits provided by the digital library component of the framework supports users and researchers in their work.

2.1.1.4 ***Open Archives Initiative***

The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) [OAI-PMH 2004] aims to allow for interoperability between repositories and services by allowing an application-independent framework for developers on which they can build their services. There exist two classes of participants in the OAI-PMH, namely:

- *Data providers*

These are the digital libraries that support the Open Archives Protocol and allows its repositories to be harvested by service providers.

- *Service providers*
  Typically, service providers do not have any content of their own and merely provide users with search functionalities that make use of data providers' repositories that are harvested using the Open Archives Protocol.

The process of harvesting metadata from data providers is done by a harvester, which is in essence an application that queries a set of repositories and collects metadata relating to the documents contained therein. Once the initial set of metadata has been harvested for all the documents, the harvester will periodically query the repositories again in order to obtain metadata for new submissions and also to update information pertaining to documents already harvested. Service providers will then use the harvested metadata to build their own search functionalities for their end-users [Shreevese 2004].

*2.1.1.5 **Current implementations***

There is a multitude of existing open source implementations of digital libraries such as EPrints and Greenstone. This section will give an overview of Eprints, Greenstone and DSpace as well as outline some of their key functionalities. In addition to open source digital libraries, there also exists a large collection of research digital libraries that provide researchers with a valuable source of research information. This section will end off with a discussion on how the features and functionalities that were identified in these examples will affect the framework.

2.1.1.5.1 **EPrints**

This discussion provides an overview of EPrints, the functionalities provided by the system as well the use of Open Access. [Eprints 2005] and [Wikipedia 2006] were used as the main sources for the information that is provided.

a) Overview
   GNU/EPrints is an open source digital library that assists researchers in publishing their work in what is called an Open Access Institutional Repository. Development began in 2000 at the Electronics and Computer Science department of the University of Southampton as an outcome of the 1999 Santa Fe meeting. An Eprints repository consists of a collection of ePrints which are pre-prints (before peer-review) and post-prints (after peer-review) of electronic research articles.

b) Open Access [BOAI 2002]

A key benefit of the EPrints system is that it complies with Open Access requirements by allowing researchers the ability to submit and review unpublished works. The Budapest Open Access Initiative has as its goal the open access to peer-reviewed research/journal articles and proposes creating a new generation of self-archiving open-access journals as a means to obtain this goal. Open Access allows researchers to publish their collections of work to open journals allowing any researcher or scholar to freely download any body of work. As a result, the research community will not only reap financial benefits but will also benefit from the accelerated research cycle [Brody 2005] and enjoy increased visibility [Hajjem 2005].

c) Functionalities [Gutteridge 2005]

- *Open Archives Initiative (OAI)*

  Supports OAI version 2.0 and supports the Dublin Core metadata schema by default and also allows users to easily configure additional metadata schemas.

- *Really Simple Syndication (RSS)*

  Users can subscribe to a repository's RSS feed which will inform them of new additions to the repository. This frees the user from having to constantly access the repository to find out if any new submissions have been made.

- *Scripting Application Programming Interface (API)*

  Provides a powerful scripting interface which allows users to write custom scripts that can be used to manage the repository based on a user's more specific needs.

- *Full-text searching*

  A full-text index of the content of the ePrints contained within the repository is maintained which allows for full-text searching. Formats that can be indexed include Adobe Acrobat, HTML and Microsoft Word Documents.

- *Multi-level browsing*

  Rather than just being able to browse by one specific field, users can create views containing a subset of documents which can then in turn be used to create new views based on a different field.

2.1.1.5.2 **Greenstone**

This discussion provides an overview of Greenstone which is a software suite that allows researchers or institutions the ability to create and distribute digital libraries.

[Greenstone 2006], [Witten 2000] and [Wikipedia 2006] were used as the main sources for the description of Greenstone's functionalities.

a) Overview

Greenstone is a suite of software that can be used to build and distribute digital library collections and was developed by the New Zealand Digital Library Project at the University of Waikato. Two key design aspects of Greenstone include platform independence and interoperability. Greenstone can run on GNU/Linux, UNIX, Apple OS-X and also Microsoft Windows. In addition to being platform independent, Greenstone is also capable of importing and exporting its data in a variety of commonly supported formats, such as METS and DSpace.

b) Functionalities

- *Data structures automation*

  Greenstone automatically creates data structures from the provided material instead of requiring users to manually sort and organize the body of information. These data structures then form the base for any searching and browsing activities done by the user.

- *OAI*

  As with EPrints, Greenstone also supports the OAI-PMH version 2 which allows it to be both a data and service provider.

- *Interfaces*

  Greenstone consists of a Reader interface which exists to allow users of the digital libraries to access information and view documents. In addition to the Reader interface, Greenstone also has a Librarian interface which assists moderators of the digital library to administer the collection of documents as well as search forms and metadata schemas.

- *Metadata formats*

  A collection of predefined metadata schemas supported by Greenstone include Dublin Core, RFC 1807, New Zealand Government Locator Service and Australian Government Locator Service. In addition to the predefined metadata schemas, moderators can also implement additional metadata schemas such METS, MARC and BibTex.

- *Available via the web and CD-ROM*

  Greenstone repositories are mainly aimed at the Internet enabled users; however, the repositories can also be distributed on CD-ROM to allow users without Internet access access to information.

2.1.1.5.3 **DSpace**

This section provides an overview as well as a description of the DSpace system's features and benefits. All information was obtained from [Dspace 2006], [Tansley 2003], [Branschofskyh 2002].

a) Overview

DSpace was developed from a joint venture, started in 2000, between MIT Labs and Hewlett Packard to assist MIT in storing and sharing their research output. After its development, DSpace was made available under a Berkeley Software Distribution (BSD)-style licence to allow other research institutions to also share documents and intellectual property among one another. A key concern which is addressed by DSpace is the need to be able to store information and documents that may otherwise be lost due to, for instance, a university department's closure or a research project ending prematurely. By providing an archiving system, DSpace allows researchers to continuously submit their research works which ensures its safe storage as well as allowing other researchers to review their unpublished works.

b) Functionalities

- *Document ingestion*

  DSpace provides the ability for administrators to import a collection of external documents into its internal storage engine. Documents are submitted as a batch where they are stored in a pool awaiting the approval or rejection from DSpace moderators. The DSpace moderators or 'gatekeepers' are notified of the new submissions after which they will review each individual document and its associated metadata for inclusion in the repository. Once a document has been accepted by a moderator, it will be visible to the users of the DSpace repositories.

- *Browsing, searching and linking*

  Users can access documents in the DSpace system using any of the following methods, viz. searching, browsing or via an external reference. Documents are stored in a logical hierarchy allowing users to drill down to the desired topic or research area using a simple browsing interface. Alternatively, if a user only has a keyword or phrase, he/she can use the search functionality to search for documents in the collection. Lastly, the document identification features of DSpace allows a user to follow an external link to directly access a specific document.

- *Interoperability*

  DSpace provides numerous methods with which it can interact with existing systems. One such feature is DSpace's ability to export data in an open XML format allowing other systems to import metadata and document content using batch operations. Moreover, DSpace supports the OAI [OAI-PMH 2004] protocol allowing it to expose its collection of documents as well as the ability to harvest documents contained within other systems.

c) Components

- *Data models*

  DSpace's data is organised in such a way as to reflect the organisation which the system serves. Each implementation is divided into the following components, viz. community, collection, item, bundle and bitstream. There exists various top-level communities, each representing, for instance, a research division such as a laboratory or department. A community comprises of various collections of similar object, for instance, the research projects with which the community is involved. Each collection contains a number of items, which are the actual documents stored in the repository. Each item is represented by a number of bitstreams, which are the actual components of the document, to form a bundle.

- *Metadata*

  Metadata is used to describe the various properties of the items contained within the various collections. The metadata is further divided into descriptive, administrative and structural schemas, each describing a different property of the item. Dublin Core is used for the descriptive metadata schema which describes information about the item, such as author, title, date and abstract. The administrative metadata is a proprietary schema which is used to describe, amongst others, the authorisation requirements associated with each item. Lastly, structural metadata is used to describe the actual presentation of the items various components, such as images and tables.

- *E-People*

  DSpace uses the term e-people to describe any person or machine which has access to its resources. E-People are represented by information such as name, surname, email address and also subscriptions. Moreover, each e-person is assigned authorisation information which determines the actions which can be performed.

- *Authorisation*
  Each action that is exposed to the e-people of the DSpace system has an associated role assigned to it. This authorisation information is used to ensure that only authorised users perform certain actions. An example role is called READ, which limits an e-person to be able to only access an item without the ability to modify that item.

- *Handles*
  A Handle is a globally unique identifier which is assigned to each collection and item contained within the DSpace communities. By using Handles to identify items, DSpace allows users to make persistent links to items which ensures link consistency. Each DSpace site is assigned a prefix which is to be used when creating a Handle for an item. A typical example of a Handle is 1721.21/367. which can be represented as hdl:1721.21/367 or http://hdl.handle.net/1721.21/367. In the former example, a user can simply enter the Handle into any capable browser and will be directed to the item on the associated DSpace site. The latter example points to a web proxy which will also direct the user to the item associated with the Handle.

2.1.1.6 ***Summary findings***

From the literature surveyed, we have learnt that a digital library allows researchers to publish their research findings to a centralised repository,which assists the research community to share their findings and ideas. The following collection of features were identified during the literature survey and have to be present in any successful digital library:

- Submit documents
- Edit, remove documents
- Metadata describing documents and repositories
- Browsing and searching repositories
- Integration between various online digital libraries
- Multiple user type support

The various properties and advantages that were identified will have to be incorporated into the digital library component of the online collaborative workspace. This will ensure that users are assisted in their various tasks when using the digital library.

2.1.2 Adaptive hypermedia

2.1.2.1 ***Brief overview***

Adaptive hypermedia systems arose from users' diverse needs regarding the presentation of information when browsing large hypermedia systems. In an adaptive hypermedia system a user can be presented with customised content and navigation links based on his user profile [Brusilovksy 1998] and [Kristofic 2005]. Questions that need to be addressed with adaptive hypermedia systems include:

- *Which systems can be adapted?* [Brusilovsky 1998]
  It is important to identify the systems or application areas where adaptive hypermedia will benefit users the most. Most commonly, these systems will contain vast amounts of information that needs to be adapted to suit the individual needs of its users.
- *What can be adapted and how?* [Brusilovsky 1998]
  Adaptation can be divided in two main categories, viz. link level adaptation and content level adaptation. The presentation of content can be tailored to suit the user's understanding of the subject matter, whilst the links can be adapted to guide the user through the information space.
- *User features needed for adaptation?* [Brusilovsky 1998], [Kristofic 2005]
  Certain information is needed in order to build a user model representing the user's understanding of a topic and specific needs. These include a user's background, navigational habits, personal preferences and goals.
- *What will adaptation achieve?* [Brusilovsky 1998]
  Lastly, it is important to determine whether the adaptation of content and links will assist the user's information and navigational needs. The goals of adaptation are specific to each area of implementation and can thus be tailored to best support the characteristics of that system.

Adaptive hypermedia systems have been implemented in a number of different disciplines, such as educational systems [Brusilovsky 1998], [Kristofic 2005], context-aware tourist guides [Cheverst 2002], [Wilkinson 2000] and online marketing [Kobsa 2001]. These systems all typically contain vast amounts of information and each user's information needs are unique, thus providing the perfect opportunity for adaptive hypermedia to support users in their information gathering needs.

Each user of an adaptive hypermedia system has his own profile which forms the basis for all personalisation functionalities. These profiles contain information relating to a user's knowledge, goals, background and preferences [Brusilovksy 1998] and [Kristofic 2005]. As mentioned before, a user can either create his personal profile by supplying certain information or his profile can be dynamically created by tracking his interaction with the site. By using a user's personal profile the hypermedia system can adapt the presentation of content, level of difficulty of the content and also the links that are available to the user. Users will thus be presented with information most appropriate for their level of understanding of a topic and can also be guided through the information space in such a manner that will best benefit their information needs.

2.1.2.2 ***User profiling/models***

As mentioned previously, each user of the adaptive hypermedia system is represented by a profile containing certain information. Some information, such as demographics [Kobsa 2001], must be supplied by the user upon registration, whereas other information can be gathered by the system by tracking a user's behaviour. Information needed to provide the user with adaptive content and navigation include [Brusilovsky 1998], [Kristofic 2005] and [Kobsa 2001],

- *User background*
  The background of the user is important, such as level of education and computer literacy. This type of information can amass to vast amounts, so careful consideration is needed when determining exactly what background information is needed. Examples of user background information include age and gender which need to be provided by the user upon registration, as it cannot be inferred from the user's actions or preferences. Moreover, a user's interests and hobbies can also be added to his background information.
- *User knowledge*
  A user's knowledge is represented by the domain knowledge that he/she has read. This information is obtained by tracking his navigational behaviour and interaction with components throughout the site. Navigational behaviour could include the links that were followed and the amount of time spent reading an article. This information can then be used to infer a user's knowledge pertaining to certain subjects and can also assist in adaptive content presentation.
- *User goals*
  By knowing the user's information needs and goals the system can anticipate the user's next actions and can also limit the scope of information to fit in with the user's goals. Typically, a user will need to supply his goals beforehand to assist

the system in quickly adapting the content and links accordingly. However, the system could also use the user's first couple of clicks to infer a goal. For instance if a user navigates to DVDs in an online shopping mall, then the system can quickly limit the scope of products when calculating recommendations.

- *User preferences*
  Preferences can be divided into preferences relating to a user's preference of a certain area of research and also when it comes to the presentation of information and links. Certain preferences must be explicitly supplied by the user, for example visual styles. However, certain preferences such as level of detail and areas of research can be inferred by the system by tracking a user's behaviour throughout the site.

2.1.2.3 ***What can be adapted***

An important question to ask about adaptive hypermedia is what can and should be adapted. One should identify the components of the system that need to be adapted for individual users. The two components of an adaptive hypermedia system that can be adapted are the content and links [Brusilovsky 1998] and [Bailey 2002].

2.1.2.3.1 **Link level adaptation**

A key component of any hypermedia system is the ability to add links allowing users to browse a hierarchy of documents and to also create links between these documents. Every user visiting a site is unique, thus his information gathering needs will be unique. This provides the perfect opportunity to adapt the presentation and availability of links found throughout the site. When working with link adaptation, the following aspects of links can be adapted:

- *Direct guidance*
  This is the simplest form of navigation support as it only relies on a simple linear navigational structure that informs the user which is the next document to view. It is best suited for educational or instructional systems where the user has to follow a linear path through the information space. In addition to providing the user with links to document that he/she must read next, direct guidance can also inform the user with documents that he/she should have read in order to understand the current set of documents.
- *Link presentation*
  A link's visibility can be increased or reduced based on its relevance to the user.

For instance, if a user has not yet followed a link and the system deems the link to be of high importance then either the size or the colour of the link can be augmented to attract the user's attention and to highlight the importance of the link. This will assist the user in deciding which link to follow whilst reading a specific document or when browsing the site's directory hierarchies.

- *Adaptive ordering*
  Another means of link adaptation is adaptive ordering where links are sorted according to their perceived importance to the user. However, adaptive ordering can cause confusion amongst users due to the fact that the ordering of links will not necessarily remain constant [Debevc 1994].
- *Link hiding*
  Probably the most helpful form of navigation support is link hiding which protects the user from irrelevant information. The system can hide links from the user if the link points to information that is not relevant to the user's preferences or if the user is not yet ready to read the information. Link hiding can be applied to contextual as well as navigational links to limit the scope of the information space available to the user.
- *Adaptive annotation*
  Links can be augmented with annotations to provide a limited amount of information to the user regarding the document to which the link points. This information can help the user to make a decision quickly on whether to follow the link or not. Examples of information include the difficulty, rating and perhaps a few keywords or a brief abstract describing the document. By providing the user with this information before following the link, the user is spared the exercise of scanning the document to determine whether it is of any importance or not.

2.1.2.3.2 **Content level adaptation**

Content level adaptation can be applied to the visual presentation of the information and physical content itself [Brusilovsky 1998], [De Bra 2003, 2] and [Boll 2003]. For example, a user could select a certain visual style as his preferred interface which will cause the information to be displayed in a certain manner. With the actual adaptation of content the following distinctions can be made:

- *Adaptive text presentation*
  A popular method of hiding information from the user is by using stretch text. Stretch text is the process of limiting the amount of information that is directly

visible to the user by replacing large sections of irrelevant or less relevant information with a single description with the option to expand the complete section. This will greatly reduce the clutter on the screen and can assist the user in quickly finding the desired information.

- *Adaptive multimedia presentation*
  Another method of adapting the content is by selectively displaying certain multimedia elements based on a user's preferences. For instance, if the user is accessing the site using a mobile device, it would be senseless to display large images and videos. The same is true for users from rural or under developed areas that have bandwidth limitations. Adaptive multimedia presentation can thus help the user achieve the most of his time browsing and reading information. Multimedia components can also be tailored, dynamically, in its style and content to suit a certain age group [Boll 2003].

- *Interface adaptability*
  In addition to the adaptation of the actual content, the interface presented to the user can be adapted to suit his specific needs. An example of such adaptation is the various visual styles that can be provided to the user to choose from. This allows users to select a visual style that is best suited to, for instance, their accessibility preferences. For example, upon registration, a user with poor eye sight could select the visual style that places emphasis on readability rather than visual richness.

2.1.2.4 ***Recommendation system***

Apart from adapting the content and links of a document, adaptive hypermedia systems can also recommend additional reading to users [Han 2005], [Schafer 2002], [Bao 2005] and [Mobasher 2001]. This is possible due to the fact that users can be clustered based on their personal preferences as well as their navigational history. These clusters of users can assist in providing recommendations based on material that similar users have read. Likewise, documents can be clustered together and recommendations can be made by informing users of documents that have similar attributes to those that a user has read previously. Dynamic recommendations can ease the amount of manual linking and suggestions made by moderators of a large collection of documents.

Recommendations can be divided into the following groups:

- *User history* [Bao 2005], [Mobasher 2001]
  By analysing a user's navigation history it is possible to recommend documents

that the user can read next or documents that the user should have read by now. Additional metadata is required in order to denote the sequence in which documents must be read. Therefore if a user has finished reading an introduction on a certain subject the system can automatically recommend the subsequent document. Likewise, if the user has failed to read an introduction to a certain subject, the system can inform the user that he/she still has to read it.

- *Profile matching against other user*s [Herlocker 2000]
  Users may be assigned to clusters based on their personal preferences, navigation history and level of interest in subjects. It can then be assumed that users in a cluster will have an overlap in their interests, therefore the system will be able to make recommendations based on documents that other users in a user's cluster have read that he/she has not yet read. In addition to providing recommendations, these groups of users can also be grouped into a community where they can share their thoughts and ideas with each other.
- *Profile matching against other documents* [Schafer 2002]
  As with the users of the system, the documents in the system can also be clustered into groups based on their attributes. If a user has already read, for instance, two documents in a cluster, he/she could automatically be made aware of other documents in the cluster that could be of interest.

2.1.2.5 ***Adaptive hypermedia methodologies***

In order to provide users with effective adaptive hypermedia services the system needs to be able to accurately model the user, the information and relationships between information items. This can be accomplished through the use of various hypermedia models, such as the Dexter Model [Halasz 1994], [Halasz 1990] and AHAM [De Bra 1999], [De Bra 2003, 2].

2.1.2.5.1 **Dexter model**

The Dexter Hypertext Model [Halasz 1994], [Halasz 1990] is the result of two small workshops that were held, the first being at the Dexter Inn in New Hampshire in 1988. A key focus of the workshops was to determine the common abstractions that were used in existing hypermedia applications. The Dexter Model divides a hypertext application into the following three layers:

- *Storage layer*
  The storage layer can be described as a database containing all of the

components and relational links between these components. A component can be described as a node containing, for instance, a document on a certain subject. One limitation of the Dexter Model is its inability to describe the structure of these components in more detail. As a result, each component is merely seen as a generic container, and doesn't allow for the distinction between, for instance, text and images.

- *Within-component layer*
  Unlike the storage layer of the Dexter Model, the within-component layer is specifically concerned with the structure of the components. The within-component layer's specifications are purposefully not elaborated on to allow for the maximum number of variations that can be stored within a component. It is up to the system's developers to choose a model that will be used to describe the structure of the components in more detail.
- *Run-time layer*
  Once all the components and relational links have been authored it is up to the run-time layer to present the content and navigation to the user. For each session instantiated by a user a new unique session ID will be assigned. Throughout a user's session the system will keep track of changes that were made to the relational links between the components. If, for instance, a component is removed or relocated in the storage-layer, all links pointing to that component will be altered accordingly. It is up to the run-time layer to handle all the interactions between the user and the system's storage and within-component layers.

2.1.2.5.2 **AHAM**

The Adaptive Hypermedia Application Model (AHAM) [De Bra 1999], [De Bra 2003, 2] is based on the Dexter Model and was developed to allow for the use of persistent user models to allow for adaptation. Adaptation is achieved through a teaching model which consists of a set of pedagogical rules that define the relationships between and sequences of documents.

In contrast to the Dexter Model, AHAM makes use of a persistent user model which is continuously updated to reflect the user's level of knowledge and navigational history. The user model can contain the history of all the nodes that a user has visited including the time spent on each of these nodes. This information can then be used to determine the level of detail in which each topic needs to be explained to the user. If, for instance, a user has read enough on a certain subject then the next time that he/she requests a page on that same subject, the system could automatically provide information that is applicable to the user's level of understanding of the subject. An example user model would be represented by:

UID (name)	***Knowledge value***	***Read***	***Ready-to-read***
AH	Expert	True	True
Digital_Library	Intermediate	False	True
OSS	Novice	False	False

Table 1: Example of AHAM user model

As can be seen in this example, a user's understanding of a subject and his readiness to read a resource is expressed in his user model. This example states that he/she has read AH and OSS but has not yet read Digital_Library. The system will allow the user to read advanced topics in the Digital_Library domain because he/she has read enough information relating to digital libraries and each time that the user has accessed such a resource the system will increase the knowledge factor which ultimately will allow the user to access the more advanced resources in the specific domain.

Together with the persistent user models, AHAM also incorporates a teaching model which uses a set of pedagogical rules for each document to determine whether a user is ready to read a certain document and also to provide users with recommendations. A pedagogical rule generally takes the form:

< Access(C) => C.read := true, post >

Illustration 1: Example AHAM pedagogical rule

In this example rule, a user's profile will be updated to set the state of resource C to read after he/she has accessed the resource.

< Access(C) and D.knowledge-value >= Known, D.ready-to-read := true, post >

Illustration 2: Example of pedagogical rule

In this example, once a user has read resource C and his understanding of subject D is at least “known”, then in the user's model resource D will be set to be ready to be read.

2.1.2.6 ***LinkBases and link services***

As mentioned in Section 2.1.2.3.1, adaptive hypermedia systems provide users with customised navigation based on their level of understanding of a subject as well as a user's personal history. In order to allow for the customised presentation of links, there exists the need for the separation of links and content. The content of the site exists in a form that contains no links, only text and images. In addition to these documents of text and images, adaptive hypermedia makes use of what is called LinkBases [Carr 2001] or link services. [Carr 1999], [Hall 1996]. Microcosm [Hall 1996] is an example of a system which allows users a personalised navigation experience by providing various LinkBases that can be applied to large bodies of information.

LinkBases allow for the separation of content and text by storing links in databases which can be queried by a system to insert links in to any standard body of text. When a user views a document, the adaptive engine will query the user's personal profile and history to determine his level of understanding of the themes contained within the document as well as the list of documents that the user has view in the past. This information can then be used to determine whether any additional links should be inserted into the document to provide the user with further reading material. Similarly, the adaptive engine could remove links which it deems are of little or no importance to the user.

Links can be stored in various formats in databases, depending on the desired use of the links. A LinkBase may contain links pertaining to a specific document or could be used to augment any document viewed by a user. When a link is intended for a specific document and word within that document, it would typically be represented in a format such as Xlink [DeRose 1989], which is a standard method of pointing to a certain word in, for instance, an HTML document. In the event that a LinkBase is intended to be used for any collection of documents, links would typically be represented by only a word and an associated URL. These links are then inserted into any document containing those specific words.

LinkBases and Link Services therefore allow for true adaptive hypermedia by a allowing content moderators the ability to separate content and links which results in users being presented with customised navigation options.

2.1.2.7 ***Current implementations***

The following section describes two implementations of adaptive hypermedia systems, viz. The GUIDE System [Cheverst 2002] and ELM-ART [Brusilovsky 1996]. An overview of each implementation as well as its functionalities and usage is provided. Lastly, the successfulness of each system's ability to provide accurate recommendations will be reviewed.

2.1.2.7.1 **The GUIDE system**

When visiting a city tourists will usually make use of a brochure or tourist map to decide which attractions to visit. Unfortunately this method of exploring a city is static and does not take into account a tourist's personal preferences, weather conditions and temporal information. For example, if it is raining a tourist will not necessarily be interested in walking around a historic castle. Instead, a tourist will rather explore the city's local galleries and museums. Adaptive Hypermedia can be used in order to circumvent these limitations. An example of such a system is the GUIDE [Cheverst 2002], [GUIDE 2006] application that acts as an adaptive guide to the city of Lancaster in England.

The GUIDE system consists of a Fujitsu TeamPad Tablet which the tourists will carry with them and a number of base stations that serves the information. When a visitor receives his GUIDE unit, he/she can supply some personal information and his personal preferences with respect to specific interests, age group, dietary requirements and attractions already visited. This information is stored on the GUIDE unit creating a personal user model which is then used when determining which attractions to recommend.

In addition to a user model, the GUIDE system also incorporates an environment model which stores information pertaining to the city's attractions. The environment model consists of geographical information, hypermedia information and active components. An active component would typically be information that can change from time to time, such as opening times of museums.

As tourists walk around the city, the GUIDE system will continuously track their movements and provide information and recommendations based on their current location. This is achieved by the GUIDE system sending a query to the nearest base station which returns information, such as opening times and activities, about the attractions in its vicinity. The GUIDE system will then use this information, together with the visitor's user model, to provide the visitor with only the relevant attractions. In addition to the visitor's personal preferences, the GUIDE system will also use temporal data, such as the time of year and day of the week, in order to further filter or adapt the information. For example, attractions that are closed on public holidays will be removed from the list and the GUIDE system could either recommend another similar attraction or provide the visitor with the dates on which the attraction is open.

Interaction with the GUIDE system is achieved through a web browser which is embedded in the application. Visitors can then read information, view images and follow links to other attractions in the city. Standard HTML is used to author the content with the addition of custom mark-up which allows authors the ability to interact with the GUIDE system. An example of the custom mark-up is the “Insert Neighbours” tag, which the GUIDE system will process and then subsequently alter the content with links to nearby attractions (based on the visitor's user model and temporal data).

On evaluating visitors' feedback of the GUIDE system, it became evident that the filtering of certain information proved to be an annoyance to the users. For example, when an attraction was found to be closed by the GUIDE system, due to for instance it being a public holiday, it was removed from the list of attractions. However, certain attractions such as museums have beautiful architecture which visitors would like to view even though they were not able to enter the building. This raises the issue of “intelligent” filtering by adaptive hypermedia applications based on a user's perceived goals versus the actual goals of the user [Suchman 1987], [Cheverst 2000].

2.1.2.7.2 **ELM-ART**

ELM-ART [Brusilovsky 1996] an online adaptive course-ware environment used to teach students the Lisp programming language. It is used in conjunction with course material and notes which is presented by lecturers to the students. The environment allows students to apply the knowledge obtained from the lectures and notes by providing example problems with real-time feedback. ELM-ART provides the following services to the students:

- *Online course material*
  Students are provided with an electronic copy of the course's material and textbooks together with a reference manual containing a set of course concepts with references to the course material. The content of the material is presented to the students as HTML pages with hierarchical links to navigate the course material and also content links allowing students to easily navigate the online course material. Contextual links between reference material are added dynamically by the ELM-ART system because it is designed to be an intelligent system which “knows” and “understands” the course material and can therefore create links between various concepts.
- *Adaptive navigation support*
  ELM-ART uses adaptive navigation to support students in their decision making when deciding which links to follow. Links are sorted according to their relevance to the student's current information needs and can be annotated with short descriptions and various icons to denote its properties. Adaptive navigation is

discussed in more detail in Section 3.3.1.6.2.

- *Prerequisite-based help*
  An additional method of assisting the student in understanding the course material is to provide him/her with recommended reading when he/she has problems understanding certain concepts. For example, if a student accesses a page which has a set of prerequisite concepts that the student does not yet understand, the system can dynamically provide links to additional material.
- *Intelligent problem solving support*
  A key feature of ELM-ART is its ability to support students with example-based programming and encourages students to re-use code from previous examples. Students are required to solve Lisp problems by entering the programming code in a text box after which the ELM-ART system will evaluate the answer and supply the student with feedback and tips if necessary. Students can also ask the system for help with solving a problem and will supply the student with advice that is based on the student's preferred method of learning.

The key to the intelligent behaviour of the ELM-ART system is its knowledge about the subject domain represented by a network of concepts, plans and rules. In support of its intelligent techniques, ELM-ART's knowledge base has been enhanced with conceptual knowledge of Lisp and the course. At the core of this knowledge base is the LISP conceptual network which represents all important concepts and relationships between them. Examples of such concepts include common Lisp objects, Lisp and general programming constructs. ELM-ART makes use of heuristics, “part-of” and “is-a” relationships to infer prerequisites and uses the resulting knowledge together with the student's knowledge of the material to adapt the content and links accordingly.

2.1.2.8 ***Summary findings***

From the preceding literature survey, adaptive hypermedia can be described as a technique that allows a system to provide each user with a customised experience when visiting a site. The following features were identified to be crucial to any adaptive hypermedia system:

- Link level adaptation
- Content level adaptation
- Customisable interfaces
- Descriptive metadata for content

- User profiling and tracking
- User preferences
- Recommendation system

After identifying the various features and properties of an adaptive hypermedia system, the next step would be to determine how each should be addressed in the development of an online collaborative workspace. Chapter 3 describes in detail how these various features were implemented in the framework in order to provide adaptive features to the users.

2.1.3 Collaborative Workspaces

2.1.3.1 ***Introduction***

In many of today's collaborative projects, members can be separated geographically from one another. This has lead to the emergence of online collaborative workspaces to allow project members to communicate with one another and to share documents and resources effectively.

2.1.3.2 ***Characteristics of collaborative workspaces***

When analysing collaborative workspaces, one finds different approaches as to how systems allow interaction between project members, resource sharing, tracking and storage. Certain systems will, for example, focus on providing members with real-time collaboration whilst others will focus more on asynchronous document management and versioning.

This section will outline some of the key features and characteristics of collaborative workspaces as they exist today.

2.1.3.2.1 **Session vs. document centric design**

Most collaborative tools fall into two main categories [Spellman 1997], [Dourish 1992], [Geyer 2003] with regard to their functionalities, namely:

- *Session-centric*

  These are synchronous systems that allow project members to communicate in real-time with one another, for example teleconferencing.
- *Document-centric*

  With document-centric systems, the focus is on document management with no synchronous interaction between project members.

Both categories have their own strengths and weaknesses. Session-centric systems are ideal for real-time communication between members. However, once the session is over there is no structured trace left of the collaborative efforts, hence there is a lack of persistence. In the case of document-centric systems, real-time collaboration does not exist, which can hamper the speed at which decisions are made. Clearly, a need exists for the existence of a system that combines the strengths of both document and session centric collaborative systems.

Spellman [Spellman 1997] has proposed such a system, and places the focus on creating a collaborative “place” where project members can meet and perform both synchronous and asynchronous work. These place-based systems enable synchronous work through the use of whiteboards, teleconferencing and chat, whilst allowing for asynchronous work through the use of document management systems. Place-based systems thus provide project members with the ability to work together on a document and then store the document allowing persistence which enables future reviews and modifications.

In addition to the need for session and document centric applications, project members also require the collaborative workspace to be easy to use [Dommel 2005]. In the past, collaborative systems' focus was on the computer and on how it can help the users, whereas the focus has to shift to the needs of the users. Users need to be able to work in the collaborative workspace and focus on their work instead of having to focus on the tools required to accomplish it.

2.1.3.2.2 **Ad hoc vs. formal systems**

Over the years collaborative workspaces have evolved because of the emergence of new technology coupled with users' evolving needs. Most collaborative systems can be grouped into either ad hoc or formal categories, with usually little overlapping between the categories [Geyer 2003], [Muller 2004].

Ad hoc systems would typically not involve more than sending documents between project members via email. This process would suffice for small scale projects where document iterations are few and the project consists of a small group of members. Ad hoc systems do, however, not provide a central repository for the tracking of changes to documents, conversation history or notes. Moreover, ad hoc systems generate a vast amount of emails, making it very difficult to manage versions of documents and organising notes.

An alternative to using an ad hoc systems is to adopt a more formal collaborative system. Formal systems have the advantage of being tailor built for collaborative work, providing users with synchronous and asynchronous communication, sophisticated document tracking and a central repository for storing resources. There are, however, drawbacks to using formal systems as they are often very complex to set up and maintain, making it difficult for project members to implement.

Geyer [Geyer 2003] proposes a new approach that is carefully balanced between ad hoc and formal systems placing the focus on sharing objects in a lightweight and informally structured system. This “object-centric” approach aims at making the objects collaboration-ware, meaning that the objects are designed to be used collaboratively instead of making collaboration an afterthought.

2.1.3.2.3 **Features of collaborative workspaces**

In order to develop an effective collaborative workspace, one first needs to understand the needs of a project's team members. In his research, Tang [Tang 1988], has identified a set of characteristics which he observed from a group of people interacting with one another during typical project meetings. These characteristics included the following:

- *Store information*
  Any documents, notes and feedback made during a meeting need to be stored to allow for future reference or modification. Online collaborative workspaces need to allow users to store resources logically so that it can be retrieved quickly and easily by the team at a later stage. A document needn't be a document in the traditional sense, rather it can be a video, animation, sound clip or application. An additional benefit provided by online systems is that members can search for resources and also compare different versions of the same document to track changes. Section 3.3.3 provides information on how a digital library full fills the need to store and retrieve documents.

- *Convey ideas*
  During a project meeting, members generate ideas such as feedback and suggestions which can take the form of small notes or illustrations. These ideas need to be conveyed to the rest of the team so that they can be evaluated and incorporated into the project. An online collaborative system should therefore allow members to share various types of objects with the rest of the team by providing a means with which a member can submit his ideas to, for instance, a shared whiteboard. In the event of asynchronous collaboration, members must be

able to submit their ideas to a shared repository where other members can view and comment on the idea at a later stage. Section 3.3.2 describes an online digital discussion forum that allows user to post and reply to a threaded discussion board.

- *Represent ideas*
  During an individual member's decision making process he/she will typically need to write down his thoughts on a piece of paper. A user must therefore also be able to store these thoughts in an online system for retrieval and expansion at a later stage. It could also be required of an online collaborative system to allow other members of the project to access and comment on these thoughts. An example of a statement database that allows users to post their thoughts and statements is described in Section 3.3.4.
- *Engage attention*
  An advantage of project members sitting around one table is that a team member can easily direct the rest of the team's attention to a specific region of a diagram or document. This is hard to mimic in most online systems due to the restrictions that word processors and other applications have. Developers of online collaborative systems must therefore implement their own methods of overlaying notes and sketches onto existing documents to allow users to direct a team's attention.
- *Manage events*
  Almost all projects have certain dates on which certain objectives need to be reached or when events are to take place. A key requirement therefore for any collaborative project to be successful is the ability to set these dates, and also to allow project members access to these dates via an online calendar. Each project can have multiple calendars, each representing a different aspect of the project, for instance, public or private events and deadlines. The IFLA-KM project, which is described in Section 4.2.3, makes use of such an online calendar.

2.1.3.3 ***Current implementations***

2.1.3.3.1 **phpGroupWare**

phpGroupWare [phpGroupware 2006] is an online collaborative environment, written in PHP, allowing project members to interact with one another using the Internet. It consists of more than 50 web based tools, such as a calendar, file manager, address book and a forum. Administrators are able to manage all aspects of the application, for example select which modules to install, set user permissions, create themes and user groups.

The ability to know each of the project members' availability and important dates are a vital part of any project. phpGroupWare includes a calendar allowing project members to schedule meetings, set recurring events, specify deadlines and also holiday periods. Project members can set email notifications for upcoming events, changes to event information and also event cancellations. Each member can also create his own categories in the calendar to help organise events. Users have the option of a set of views when exploring the calendar, for example day, month, year views, as well as a group planner displaying the user's various categories with a detailed time line of all events.

phpGroupWare has a fully threaded forum allowing project members to create topics, post messages and reply to existing messages. Each aspect of the project can therefore be represented by a topic with its own threaded discussion reducing the amount of time required by users to find the relevant threads. One feature that phpGroupWare's forum lacks is the ability to search for messages in the forum.

A file manager provides a central repository for storing files relating to the project, allowing users to easily share documents. The file manager does not, however, allow for any metadata describing the documents, limiting the ability to properly describe each document and complicating the process of keeping track of versions.

phpGroupWare has a built-in RSS reader which allows project members to keep track of important news events. Project leaders can prescribe a set of required RSS feeds, whilst users can select additional feeds that match their personal preferences. Each RSS headline links to the detailed news item or article, providing a very simple and effective method of conveying essential information to project members in a central location.

Another method of sharing useful and important information with the team is through the use of shared bookmarks and notes. A bookmark consists of the URL, title, description and rating for the resource and is stored in a hierarchy of bookmarks according to its subject. In addition to the bookmarks, the notes system is also a useful method of sharing bits of information between project members.

phpGroupWare is an excellent example of an open source implementation of an online collaborative workspace. It supports almost of all the required functionalities outlined by Tang [Tang 1988], with the exception of a shared whiteboard and the ability to properly support document versioning.

2.1.3.4 ***Summary findings***

A collaborative workspace allows researchers to share their thoughts and ideas with one another during a project's various stages. This allows researchers to always have access to each other's thoughts and inputs. Collaborative workspaces comprise the following features and characteristics:

- Session vs. document centric designs
- Ad hoc. vs. formal systems
- Store information
- Convey ideas
- Represent ideas
- Engage attention
- Manage events

It is important to keep all of these features in mind when developing an online collaborative workspace, as the integration of the various components will determine the success of the framework. The framework will therefore have to integrate the various components to ensure that users can seamlessly move between each component whilst performing their tasks.

2.2 ***Open Source Software***

As is stated on The Open Source Initiative's website [OSI 2006], Open Source Software is "The **basic idea behind open source** is very simple: When programmers can read, redistribute, and modify the source code for a piece of software, the software evolves. People improve it, people adapt it, people fix bugs. And this can happen at a speed that, if one is used to the slow pace of conventional software development, seems astonishing."

2.2.1 A brief history of free/open source software

The following history of free/open source software has been adapted from [OSI 2000], [GNU 2006].

In the 1960s and 1970s computer science students would freely distribute the tools that they developed in university laboratories amongst one another. This resulted in the idea that software should be free and open to allow a community to easily make modifications and to share those modifications with the rest of the community. Richard

Stallman started his computer science career in 1971 at the height of this software sharing environment, which prompted him to develop a vast array of freely available tools for the ITS operating system on which he worked.

In the early 80's, this community of students broke down with discontinuation of the PDP-10, for which the ITS was developed, and the subsequent emergence of the proprietary VAX operating system. This resulted in the students being lured away from the universities' research laboratories to the commercial world of the closed source proprietary software model.

Stallman was determined to continue the free software community in which he started his computer science studies, so he decided to devote himself to creating free software. Free software, according to Stallman, must allow users the following:

1. *Run the program for any purpose*
   Users must be granted the right to use the software for any purpose as they see fit without any restrictions.
2. *Modify the program*
   Each user of a piece of software has unique needs, therefore users must be able to alter the original software to suit their needs. Moreover, they must be allowed to redistribute their modified version of the software free of charge or for a fee, in the case of additional support being provided, as they see fit.
3. *Redistribute copies*
   Once a user has obtained a copy of the software, he/she must be allowed to share it with his friend or post it on for instance his web site to allow users to download it for free.
4. *Distribute modified copies*
   Modified versions of the original software must also be allowed to be distributed freely to allow the rest of the community to benefit from the enhancements to the software.

In 1984, Stallman resigned from MIT and formed the Free Software Foundation to support his and other developers' efforts in creating free software and focussed his efforts on his GNU (Gnu's Not Unix) version of the Unix operating system. In addition to Stallman's work, the X Consortium created a free windowing system and Larry Wall created Perl, which is one of the most commonly used scripting languages to date.

The GNU General Public License (GPL) was drafted in order to ensure that free software would always remain freely modifiable and distributable. The GPL states that the source code of free software may be viewed, changed and added to, as long as it is then again distributed under the same GPL license.

In 1991 Stallman's team started working on their own kernel for their implementation of Unix called GNU. While development was under way on the GNU kernel, Linus Trovalds started his own implements of the Minix kernel which he called Linux. The Linux kernel was quickly adopted by the free software community and programmers were allowed to modify the original source and resubmit it back to Linus. In no time at all Linux became the *de facto* kernel for the open source community.

One of the landmark events in the history of free software was when, in 1998, Netscape decided to release the source code of their web browser under the GPL. Shortly after Netscape's announcement, a body of individuals formed the Open Source Initiative (OSI) to highlight the benefits of free software to the business community and to certify free source licenses. This, together with Netscape's announcement, prompted other large companies such as IBM, Oracle and Corel to follow suit in their support of Linux.

The use of open source software has grown at a tremendous rate, for instance Linux's share in the server market has since 1997 grown from a statistically insignificant percentage to 28.3% [ZDNet 2004]. This is a clear indication of the popularity and advantages obtained from the use of free/open source software.

2.2.2 The open source model

Eric Steven Raymod, in 1997, wrote a book entitled “The Cathedral and the Bazaar” [Raymond 2002] in which he reviews two development models used in the software environment today. In his book, Raymod describes what he calls the Cathedral and the Bazaar development models and provides a case study of an open source utility which was developed using the Bazaar model. Traditionally, large projects would be based on the Cathedral model where there exists clear development goals and each task is specifically assigned to an individual or group of developers. In contrast, the Bazaar model allows for projects to evolve organically and allows any developer to take part in the development and testing of any component of the project. Most projects in the Linux world have been developed using the Bazaar model, which has greatly contributed to the speed and frequency with which modifications and updates are made.

The Bazaar model does not clearly identify roles and goals and should exhibit the following patterns [Robles 2004]:

- *Users should be treated as co-developers*
  This allows the development process to be opened to the maximum extent

allowing users to take part in testing, suggesting and actual development of the project.

- *Early releases*
  A project's initial release should be made available to the public as soon as possible to allow users to become co-developers in the project. This will greatly increase the interest and impact of the project.
- *Frequent integration / Release often*
  By frequently releasing updates to the project, it is allowed to evolve in an incremental way, at the same time allowing for parallel debugging and frequent feedback from users and developers.
- *Maintain several versions*
  Usually, there will exist two versions of a project, viz. the development version which is less stable but contains the latest features and a production version which is more stable but is less feature rich. This allows for both developers and more conservative users to use the software whilst satisfying their individual needs.
- *High modularization*
  Central to the success of the Bazaar model is the reuse of modules by different software projects, which limits the need to code certain features from scratch.
- *Dynamic decision making structure*
  The decision making process with open source projects can range from a single leader who delegates responsibilities to other developers, a more democratic model where decisions are made together or committees that represent whole families of projects.

Most open source projects, such as Linux, Netscape, Apache and Perl, are developed using the Bazaar model. Moreover, SourceForge [Sourceforge 2006] contains more than 110,000 open source projects, all of which are being developed using Raymond's Bazaar model.

2.2.3 Open source development tools

A vast array of development tools have emerged together with the rise of open source projects. These are essential tools needed to guarantee the success of any large scale open source project. Included in the list of tools are:

- *Source code version control*

Most open source project developers are volunteers and do not necessarily work in the same office or even continent. This has resulted in the need for a robust version control system that can allow multiple developers to work on the same file simultaneously. An example of such a system is Concurrent Versioning System (CVS) [CVS 2006], which allows for concurrent development on a file and the subsequent merger of all the changes.

- *Testing tools*
  Testing tools are used when systems undergo integration, and is used to provide automated testing. An example of such a system is Tinderbox [Tinderbox 2006], which assists developers by running a continuous build of the project and informs developers of any integration problems.
- *Bug tracking tools*
  Central to the success of open source projects is the ability to quickly identify, report and fix errors in the code. These tools are used to log bugs and the current status of the bug, who is responsible for the bug and common scenarios in which the bug might occur. It is then up to the author of the offending bug to fix it and update the status of the bug and submit the updated version to, for example, CVS or Subversion. Examples of such systems include Bugzilla [Bugzilla 2006] and GNU GNATS [GNATS 2006].
- *Communication*
  Developers of open source projects are usually spread across the globe making it necessary for tools to allow for communication between one another. This is accomplished via websites, such as Freshmeat [Freshmeat 2006], SourceForge [Sourceforge 2006] and GNA [GNA 2006], mailing lists and instant messengers.

2.2.4 Open source projects

Open source has been embraced by a large number of companies and user groups. However, there are some that stand out as being at the forefront of bringing open source software to the world. These include the Apache Software Foundation [Apache 2006], Linux [Linux 2006], [Kernel 2006], MySQL [MySQL 2006] and the Mozilla Foundation [Mozilla 2006].

Section 3.2.3 describes the open source software that was used in the development of the framework described in the dissertation.

2.2.4.1 ***The Apache Software Foundation***

Apache is one of the most recognisable names in the open source community, most notably for its web server. The Apache Software Foundation (ASF) [Apache 2006] was formed in 1999 as a natural outgrowth of The Apache Group, who, in 1995, was responsible for the development of the Apache HTTP Server [Httpd 2006]. The primary goals behind the ASF is to:

- Provide a foundation for open source projects by providing hardware, communication and business infrastructure.
- Distribute funds that were received to best support the development of its projects.
- Create a legal body that protects its volunteers from any legal actions.
- Protect the Apache brand, applied to its projects, from being abused by other organizations.

From 1995 until 1999 a group of developers came together to maintain the web server that was created by the NSCA, which was abandoned by its original developers. That server was freely available and the source code was also available for anyone to download and modify. As a result of the openness of the project, users were able to start maintaining the project themselves and quickly started to exchange patches and information on how to prevent problems and to enhance the current software. The web server that was developed from 1995 until 1999 became known as the Apache HTTP Server and accounts for 65% of the server market today. In addition to their web server, the ASF is now also responsible for a vast number of other projects such as mod_perl [mod_perl 2006], Xerces [Xerces 2006] and Jakarta [Jakarta 2006].

The success of the ASF is due to the manner in which projects are managed. Each project is lead by a committee supervising all aspects of the project, such as its users, developers and committers. A user is someone who uses the software developed by Apache and might contribute by providing feedback and reporting on new bugs that were discovered. In contrast to users, developers are programmers who submit patches, write documentation and suggest features or enhancements. Developers are encouraged to actively participate in the development of the project by the prospects of becoming a committer. A committer is someone who also actively develops but has the added privilege of being allowed write access to the code repositories, thus not having to wait for someone else to provide a patch or suggestion.

2.2.4.2 ***Linux***

Linux [Linux 2006], [Kernel 2006] is usually used to refer to the kernel and the complete operating system and is one of the most prominent examples of free software. In 1991, Linus Trovalds began working on a clone of the Minix operating system at the University of Helsinki. His aim was to extend the original Minix kernel but was denied access to the original source code, forcing him to develop his new kernel from scratch. The first version of the kernel, version 0.01, was released on the Internet in September of 1991 and was followed in October by version 0.02. While Linus Trovalds was developing his kernel, Richard Stallman (founder of the GNU movement) was also busy developing a kernel for his set of GNU applications. Stallman eventually gave up on his kernel which resulted in the “marriage” of Linux and GNU.

Officially, the term Linux is to be used for the kernel which Linus Trovalds developed and GNU/Linux for the combination of the set of GNU tools and the Linux kernel. GNU/Linux, by itself, is rarely used on its own and is usually packaged in distribution or distro. A distro uses GNU/Linux together with the X-Windowing System and a custom installer to allow users to easily install and interact with the operating system. Currently there exists roughly 400 Linux distributions of which Red Hat [Red Hat 2006], Debian [Debian 2006] and Slackware [Slackware 2006] are the most common, to name but a few.

The Linux kernel, as with GNU applications, is contributed to by a vast number of developers. Linus Tovalds, however, has the final say with regards to the direction in which the kernel must go and also which features and patches are allowed into the official version of the kernel. Typically, when a developer would like a new feature, he/she would have to write the code himself, or get another kernel developer to do it. Once the feature has been compiled, it will be tested thoroughly by numerous developers, after which it would be submitted to Linus Trovalds, where he will decide whether to allow it into his source tree for the official kernel. This development model allows for a great number of contributors, greater stability whilst ensuring that Linux has a clear direction for its future.

2.2.4.3 ***The Mozilla Foundation***

The Mozilla Foundation [Mozilla 2006] has much the same roots as the Apache Software Foundation (ASF). As with the ASF, the Mozilla Foundation was formed as a result of their flagship project; the Mozilla web browser. In 2003, Mozilla.org, which coordinated the Mozilla open source development, announced the formation of the non-profit Mozilla Foundation aimed at continuing Mozilla's open source development projects. America Online played a vital role in the formation of the foundation by providing start-up funds and additional resources such as equipment, trademarks and intellectual property.

A distinction must be made, however, between Mozilla and the Mozilla Foundation. The Mozilla Foundation is the non-profit organization created to support Mozilla in its open source development projects. Mozilla has as its key characteristics the following:

- *Software producer*
  Mozilla is best known for its Firefox web browser [Firefox 2006] and Thunderbird email client [Thunderbird 2006].
- *Provider of development tools*
  In assisting open source development, Mozilla has created an error logging web tool [Bugzilla 2006], a web tool that allows for querying of a project's source files [Bonsai 2006] and a web tool that constantly reports on the various builds of their applications [Tinderbox 2006].
- *Open source community*
  Founded by Netscape in 1998, Mozilla quickly became a leader in the development of open source projects. Mozilla's millions of users and large development community can be attributed to their use of the Internet to connect developers with one another and also to keep in touch with their users' needs. The Mozilla developer community consists of forums, newsgroups, chat rooms and developer zones containing documentation on its more than 150 projects. There also exists more than twenty Mozilla related web sites aimed at Mozilla's user community with information ranging from simple user guides to tutorials on how to extend Mozilla to suit a user's specific needs.
- *Technology industry partner*
  Over the years Mozilla has partnered with numerous large corporations such as IBM [IBM 2006], Red Hat [Red Hat 2006] and Sun Microsystems [Sun 2006, 2] while many large institutions, such as the French Ministry of Agriculture, have switched to Mozilla's software. Mozilla's also provides resources on how to implement their software in third-party applications.
- *Advocate of standards*
  Mozilla is a great advocate of web standards and strives to create software that complies the standards set out by the World Wide Web Consortium (W3C) [W3C 2006].
- *Educators*
  In order to promote the use of its technologies and also web standards, Mozilla

has compiled documentation aimed at software developers, web developers and also end users. This documentation assists developers in understanding web standards and also provides then necessary information on how to successfully implement Mozilla's software.

Mozilla has played an integral part in the promotion, development and guidance of open source software over the past couple of years and will continue in doing so in the future.

2.2.5 Motivation for the use of open source software

As stated previously, open source software was used for the development of this framework because of its various benefits over proprietary software. This section will expand on the various benefits of open source software, as compiled from [Gonzalez-Barahona 2000], [Bobulous 2004], [CPR 2004].

- *It's free*
  Most research institutions would rather spend their resources on research instead of software. For this reason, the use of open source software, which is provided free of charge, is much more beneficial to research institutions with limited financial resources. Open source software, therefore, allows institutions the opportunity to obtain sophisticated solutions without the cost incurred when opting for proprietary solutions.
- *Availability of source code*
  The source code of open source software is available to anyone and can therefore be altered to suit a specific user's or institution's specific needs. In the event that a certain feature is not provided by the original software solution, users then have the choice to either implement the feature themselves or obtain a modified version of the software. This allows users and institutions much more freedom when compared to proprietary software.
- *Cross platform*
  The ability to modify open source software allows developers to port software to any operating system. This allows, for instance, Apache's web server to run on Linux, Unix, Apple's Mac OS and Microsoft Windows. By providing users the freedom of choice with respect to operating environments, open source software limits the need for additional supporting software to be acquired. For example, if an institution uses a Unix server and a software solutions used MySQL and Apache and was developed on Mac OS, it can easily be ported to Unix without the need to purchase an additional server running Mac OS.

- *Not bound to a specific company*
  Open source software is usually developed by a group or community of developers who act on a volunteer basis. For this reason, the life time of open source software is usually not bound to a specific company or individual. Another advantage of the community based development is that the interests of users are pursued instead of ideals or marketing objectives of a large company. This ensures that the software will always focus on supporting users and their requirements as well as international standards.
- *Additional choice*
  Open source allows users much more choice within a product or solution. One such reason is the ability of open source projects to “fork”, creating a new project with a new set of goals and features. A project might fork because of developers' different views on the direction of the project or in the case where the target users might differ. An example of where the target users base might differ is with stable and experimental releases of a project. Certain users might prefer a more stable production release of a solution, where another group could prefer an experimental release for use in research environments. Moreover, the turnaround time for additional features are much quicker than that of proprietary software, allowing users easier access to more feature rich software.

The motivation for the use of open source software becomes clear from the above mentioned list of advantages. As a result, the final framework will be more accessible to researchers and users because of its cost, portability and availability. In the end, researchers and institutions will benefit from the various advantages of open source software.

2.3 ***Usability***

Usability is concerned with the ability in which a system succeeds in addressing the human computer interaction requirements. A project or institution would greatly benefit if the following list of usability attributes, identified by Nielsen [Nielsen 1993], are addressed.

- *Learnability*
  The framework was developed with the user's needs in mind, and therefore the need to make the system easy to learn was critical. By constantly consulting users and project stakeholders, each project implementation could be tailored to suit the proficiency level of the target users. This allowed the framework to be

quickly adopted by users allowing them to rapidly learn the various features at their disposal.

- *Efficiency*
  It is important for a user to be able to quickly perform a set of tasks after he/she has become familiar with the available features. Once again, by constantly consulting the various stakeholders, the implementation of the framework could be tailored to provide the users with the necessary features to allow them to effectively perform various tasks. For example, certain metadata fields could be populated automatically for a user when he/she submits a document to one of the digital libraries. This frees the user from having to repeatedly enter information, such as the name and date of a conference, when submitting conference papers.
- *Memorability*
  A user must retain his skills once a user has learned the features of the system and has become proficient in performing certain tasks. The memorability of the system would therefore allow any user who has not used the system for a certain amount of time to easily continue to perform certain tasks again when needed. Once again, this is achieved by consulting the various stakeholders and ensuring that the process of performing tasks are simple and easy to remember.
- *Low error rate*
  A low error rate will greatly increase the efficiency of users as well as ensure that the system satisfies their needs. It was therefore important to observe the users when they performed certain tasks such as submitting a new document to the digital library. A clear understanding of common mistakes and pitfalls were identified by observing users and the common mistakes that they made whilst using the system. This information was then used during the implementation of each instance of the framework to ensure a low error rate. For example, certain fields are required when submitting a new document to the digital library. In order to ensure accuracy, these required fields were first validated to limit the number of times that a user had to go back and fix errors.
- *Satisfaction*
  Users will be encouraged to use a system more often when they enjoy the tasks that they need to perform. It is therefore important that the system be developed and presented in such a way that the user experiences a high level of satisfaction. This high level of satisfaction can, as with the other usability requirements, only be obtained when the input from users are evaluated and

incorporated into the system. For example, the interface plays a large role in how the user perceives the system, emphasising the need to tailor the design and visual elements to meet the requirements of the target users.

*2.4 **Literature survey findings***

This section will outline the findings of the literature survey and will first provide a reflection on which components and features can be integrated in order to develop a modular system that complies with user requirements. Thereafter, the role of open source software in the development of such a framework will be discussed.

2.4.1 Critical reflection on the literature

The preceding literature survey provided a better understanding as to which components need to be implemented in an online collaborative workspace as well as how each of these components should be implemented. Following is a list of the main components that were identified:

- Digital library
- Adaptive hypermedia
- Collaborative workspace

It is important to integrate the various components and their respective features, that were identified in the literature survey, in order to develop a framework that meets all the user requirements. From literature survey it can be seen that there exist overlapping features between the various components which can be integrated to provide an online collaborative workspace that supports researchers in their research efforts. The features that can be integrated to provide a truly online collaborative workspace include:

a) Metadata
The use of metadata is critical when describing information objects contained in both a digital library and an adaptive hypermedia system. Metadata can be used to describe the various properties of objects, ranging from title, author and abstract to more complex properties such as level of difficulty, sequencing and requirements. It is therefore important to find a unified metadata solution that can be integrated for all the information objects used throughout the framework.

b) Information management
The submission and management of information objects is a common occurrence

in all three of the main components. Each component, however, has a slightly different approach to storing and managing information objects. It is therefore important to find a means of managing these information objects that is best suited to all scenarios. A unified way of submitting, editing and revising these information objects would therefore greatly assist the development and modularity of a framework supporting these various components.

c) Information retrieval
In addition to the management of documents, it is also necessary for users and researchers alike to be able to retrieve these information objects using various information discovery techniques. These techniques include browsing repository hierarchies, searching and direct linking. Certain techniques are more suited to other components due to the nature on the information objects. For example, research documents are described by a richer metadata schema, allowing for more complex search queries to be performed.

d) Convey ideas
An important aspect of researchers working in a collaborative environment is the ability to convey ideas relating to various topics. These ideas can take various forms, ranging from simple statements to threaded discussions on certain topics. Examples of such components include threaded discussion boards, synchronous chat applications and shared whiteboards. Once again, metadata helps with both the organisation and retrieval of these ideas.

e) User profiling
User profiles can be used to provide users with customised information as well as recommendations. This can be useful in projects where researchers focus on one specific area in a multi-dimensional research project. By using profiles, users are spared the time consuming task of filtering through information that is of no importance to their specific domain. The use of metadata to both describe user interests and information objects is invaluable when determining which information is best suited for each user.

By reflecting on the various components that were identified in the literature survey, it is evident that the integration of these components will provide a framework that would meet all the user requirements of an online collaborative workspace. A framework comprising of a digital library, collaborative workspace and discussion forum would allow users to share their thoughts and research findings in one centralised place, which will greatly improve efficiency. Effectiveness is achieved through the standardisation of information management by both the system and the users. Moreover, users will become accustomed to performing tasks in one system, using standardised steps, which will allow them to quickly become experts in the submission as well as retrieval of information.

The following chapter details the various modules that were identified to comprise the framework. These modules were derived from the components and features that were identified during the literature survey. Following is a list of the core modules that were developed to meet the user requirements of an online collaborative workspace.

- *Content module*
  One of the features that were identified to allow users to focus on a certain area of a research project is adaptive hypermedia. The content module provides the framework with adaptive hypermedia capabilities to customise the content, presentation and navigation for each individual user.
- *Online discussion forum module*
  An important aspect of an online collaborative workspace the the ability for researchers to share and convey ideas. The online collaborative workspace provides users with a threaded discussion forum where they can post, read and reply to comments and ideas.
- *Digital library and workspace module*
  Most research projects have as an outcome the publication of various research findings. The digital library and workspace allows researchers to publish the documents and researcher to a centralised repository where it can then be accessed by the research community. In addition to providing researchers with a centralised repository, the workspace also allows researchers to review and revise a document during its life cycle.
- *Statement database module*
  The statement database allows researchers to make comments regarding certain topics or ideas. It therefore provides researchers with another form of conveying ideas to the research community.

2.4.2 Open Source Software

Open source software allows developers the ability to develop applications and frameworks using software that is freely available. This allows institutions to rather focus their resources on research instead of expensive proprietary software. The various software and programming languages that were reviewed include the following:

- *Programming languages*

  There exists various programming languages that can be utilised in order to achieve a certain goal. Java is an example of a programming language that can be used in both the development of standalone applications as well as sever-side programming. The best example of where Java is used as a server-side language is in Java Server Pages, which allows for very robust and sophisticated web-based applications. Another language that can be used in both a scripting as well as server-side environment, is Perl. Perl is a scripting language that is renowned for its ability to process data such as text files. An extension to Perl, which was developed to enhance its server-side performance is mod_perl.

- *Mark-up languages*

  Mark-up languages have come a long way in the past couple of years. An example of how these languages have evolved is the use of XML which can be used to describe the complex properties of documents and information objects. XML has the advantage that it is readable by both humans and computers and its ability to be used in a multitude of programming environments. A companion to XML is XSL, which is used in the presentation of XML data structures. The combination of XML and XSL, therefore, allows for multiple views of the same XML data.

- *Databases*

  A relational database provides the ideal storage mechanism for data, because of its ability to effectively store, alter and retrieve information. MySQL is one of the best known open source databases in use today. Its conformance to standards, speed, scalability and portability makes it an ideal choice in any development project. SQL is a standard language which is used to access and alter data contained within a relational database. The advantage of SQL is that it is, as with XML, readable by both humans and computers.

- *Servers and operating systems*

  A web server is needed to allow users access to server-side applications, such as CGI and JSP. Two well known open source examples of such servers are Apache's web server and Jakarta Tomcat. These two servers can be used in unison to lever the benefits of each. A typical configuration would be to use Apache to serve static content and CGI whilst forwarding any request for JSP to Tomcat for processing. Linux is an operating system that is freely available and is perfectly suited to be both a workstation as well as a web server.

2.5 ***Summary***

The preceding chapter provided a review of the various components and methodologies which can be used in the development of an open source online collaborative workspace. Included in the review of components were digital libraries, collaborative workspaces and adaptive hypermedia. A clear understanding of each component's roles and advantages was obtained which greatly assists in the development of an online collaborative workspace.

The components that were identified satisfied one of the two main objectives that were set out in the research question. Developing an open source framework is the other main objective and the review of the open source movement, development methodologies and technologies provided valuable insight into how to accomplish this objective.

Following the review of the components and tools required to develop an online collaborative workspace, the next chapter now describes the development process of such a system. The chapter starts with a review of the development methodologies that were used in the development of such a framework followed by the main components and features.

3 Chapter 3 - Framework architecture and research methods

3.1 Introduction

When designing a framework of this size and nature, it is important to consider various design strategies, architectures and technologies in order to provide the best solution. The various strategies that can be followed could have a great impact on the development style, the ability of the framework to adapt to changes and also the ability to make changes in the future. This chapter describes the approach that was taken in order to make the framework as flexible, robust and reusable as possible, whilst using the best open source software available.

The main specifications of the framework were derived from the various functionalities found in the online collaborative systems which were discussed in the previous chapter. These include, amongst others, an online digital library, discussion forum and user management. In addition to the components and functionalities that are found in most online collaborative systems, this framework also contains an adaptive component which allows for the customisation of content and presentation to each user. Each of the functionalities were closely examined to determine how exactly they would be implemented and integrated to allow for modularity of the framework.

This chapter therefore consists of the following three main sections,

- *Design principles*
  The design principles of the framework guided the development process by providing a sound basis for the work to be done. There were three main principles which were followed during the development stages, viz.
  - Modularity
  - Three-tiered architecture
  - Open source software
- *Functional components*
  The function al components consist of four main modules which aim at addressing the various user requirements that were identified during the literature review, and consist of:
  - Content module
    As mentioned previously, adaptive abilities were added to the framework to

allow users personalised presentation and content [Brusilovsky 1998], [Kristofic 2005] based on their unique needs and interests. This module is responsible for the presentation of content that is tailored for the user based on his personal profile. Each page contained within the module has specific mark-up to allow sections for be expanded, recommendations to be made and link alterations which results in each user being presented with a custom view of the page.

- Discussion forum module

  The online discussion forum allows user who might be based in various locations around the world to exchange ideas and comments. Each branch of a project can be represented by a different topic in the discussion forum allowing project members to subscribe to only those areas of the forum which are of interest to them. The discussion forum's threaded approach provides users with the ability to reply on entries which allows other users to, in turn, reply one those replies.

- Repository and workspace module

  A key requirement of any collaborative project is the ability to share documents and ideas [Anderson 1997], [Wiederhold 1995]. The repository assists project leaders in storing document for a project and allows users access to a structured hierarchical view of the repositories. In addition to providing users with documents, the repository module can also be extended to become a workspace where users can edit and submit multiple versions of a document. Moreover, users can search the documents in the repositories making use of any of the three search interfaces, viz. simple, advanced and full text. Each document in the repository is described using the full Dublin Core metadata schema as well as a collection of additional metadata fields developed specifically for the framework.

- Statement database module

  The statement database provided a space where users can make a statement pertaining to certain topics. A statement can range from a quote to a mission statement to a question. Unlike the discussion forum, the statement database is therefore a space where users can share their views for which they do not expect a reply.

* *Core supporting modules*

  - Graphical user interface

A key aspect of the framework was to allow for the modification of the interface without altering the underlying Java code. An advantage of this separation is the customisation which can be achieved in order to suite a user's personal preferences.

- User component
  A key functionality of the framework is the ability to build a user profile that describes a user's individual interests and contains personal information. Each user is represented by a User component containing his personal details, history and personal preferences.
- Plugins
  Plugins are used by the framework to allow for the customisation of certain events that are triggered by modules and components. Each module exposes a set of events that can accommodate the use of custom plugins.
- Security
  Security plays a key role in the development of the framework to ensure authorization and authentication of users and their actions throughout the site. A role-based security model, that contains five levels, is used as the base for the security of the framework.

3.2 ***Design principles***

Various design principles were followed throughout the development of this framework. These principles ensured that the resulting framework would be robust, scalable, flexible and customisable to suit the various requirements posed by each implementation. This section will discuss the three main design principles and how each attributed to the framework's success. Section 5.3.3 in chapter 5 reflects on the benefits that were identified from the use of the various design principles.

3.2.1 Modularity

The framework is built on a modular architecture in order to allow for the addition, removal and modification of modules without impacting the entire system. Each module was developed with careful consideration for the rest of the framework to ensure optimum code reuse and allow for integration between each of the modules. When a new module is added to the framework, it can then make use of the Application Programming Interfaces (API) exposed by the existing modules which greatly increases effectiveness and productivity.

3.2.2 Three-tiered architecture

In addition to the modular approach, the framework also makes use of a three-tiered architecture [CMU 2006], [Wiederhold 1995], [Halasz 1994], [Halasz 1990] allowing for the separation of the database, processing and presentation layers. This separation increases performance, flexibility, reliability and re-usability in a system. The three-tiered architecture comprises of the following layers:

- *Database*
  A relational database is used to store all the information contained within the framework, such as users, discussion forum posts and document metadata. Standard SQL queries are used by the business logic layer to interact with the database when updating, inserting and removing data.
- *Process management/business logic*
  This layer is responsible for performing process management and interacting with the database and presentation layers. Typical operations performed by this layer includes database queries, instantiation of components and logging of system processes. All of the components mentioned in this chapter are stored in this layer.
- *Presentation layer*
  The presentation layer is only layer of the architecture that the user actually interacts with. It is responsible for communicating with the correct objects middle tier and also for rendering the resulting output obtained from those objects.

3.2.3 Implemented technologies and software

Open source software was used exclusively in the development of the framework in order to make it as accessible, flexible and cost effective as possible. A great number of technologies and software applications are used by this framework, each contributing to either a certain area of development or for providing a platform on which the framework is hosted. There exists three main categories into which the software and technologies can be grouped, viz.

- *Helper scripts*
  These scripts are not part of the core framework but eases certain cumbersome tasks such as installing and configuring a new instantiation of the framework. All of the scripts are written in the Perl [perl.com 2006], [perl.org 2006], [CPAN 2006] programming language and executed on the command line of the operating system.

- *Core functionalities*

  These are the core functions and components out of which the framework is composed. Java [Sun 2006] is used as the primary programming language for the middle tier whilst a combination of XML [xml.org 2006], [xml.com 2006], XSL and JavaScript [Javascript 2006], [WDVL 2006] Is used for the presentation layer. The front-end of the framework, which is used to display everything to the user, uses Java Server Pages and CGI.

- *Server software*

  The server software provides a platform on which the framework is hosted and consists of a MySQL [MySQL 2006] database and Apache HTTP web server [Httpd 2006]. It was decided to make use of open source solutions based on the fact that proprietary software, such as Microsoft's Internet Information Services and SQL Server, would have added a considerable financial overhead to the project.

3.3 ***Functional components***

The literature survey and informal interviews provided insight as to which components and features should be implemented in an online collaborative workspace. This section will describe, in detail, the various modules that were developed to provide a framework that meets satisfies the user requirements that were identified. Each module will be described by a section containing an overview, the various classes comprising the module, its functionalities, integration into the core framework as well as its configuration.

3.3.1 Content module

3.3.1.1 ***Overview***

The content module provides for either static or dynamic content. In addition to providing users with content, the presentation of the content and links can also be adapted to suit the preferences and knowledge level of the current user. This is known as Adaptive Hypermedia and is discussed further in Section 2.1.2. It has to be stated here that there is a difference between the documents contained within the repositories and the static and dynamic pages of the Content module. Section 3.3.3 describes the repositories and their documents in more detail.

Static content per definition is content that is supplied to the system that requires no special processing or adaptation. An example of static content would be a page that was copied from a journal or a newsletter that is to be displayed to all users in the same manner.

Dynamic content, on the other hand, allows for adaptation to be presented in a unique manner to each user and also to the same user over time. The main principle behind dynamic content is to adapt the presentation of content and links to suit the current user's needs. For instance, certain sections might be hidden to a user if the system deems it necessary not to display the content again.

Another feature of the content module is that it allows for the cross-referencing of pages within the collection to allow for a fully linked hyperspace. Pages can be linked using either of the following methods;

- *Linking*
  Links are created within the pages that can reference either another page within the collection or any other resource on the Internet or within the site.
- *Sequencing and requirements*
  When dealing with educational or instructional pages, certain pages might have a set of requirements that need to be met before the content can be understood. Thus, by identifying required pages the collection can become interlinked allowing a user to further explore his knowledge of a topic.

3.3.1.2 ***Classes***

In order to develop the content module the following set of classes were developed.

3.3.1.2.1 **Module**

The module class is found throughout the framework, and is essentially a specialization of the root Module class that defines and exposes certain required functionalities. Most of the required functionalities have already been defined in the root Module class, thus requiring very little specialization of the class.

An example of specializing the module is the loading of security roles associated with each of the subjects. Each subject contained within the site has its own set of required security constraints, for instance, submitting a new page requires a user to be at least a WORKER.

3.3.1.2.2 **Loader**

The loader is also a specialization of the root Loader class and is primarily concerned with loading initial configuration values, such as subjects, from the configuration files. A thread runs in the background to assure that any changes made to the module's associated configuration information is picked up and handled accordingly.

3.3.1.2.3 **Metadata**

Metadata is used to describe a page as well as each section and link within the page. In essence, the metadata class consists of a collection of fields that can be used to describe a resource. The field information is loaded for the module's configuration file and stored in a hash table to allow for lookups. Each metadata field is an instance of the MetaDataField class, which will be further described in Section 3.3.3.2.3.

In addition to containing a list of all the fields, the metadata class also assists in ensuring that required information is supplied when submitting a new page by utilizing the Validator component.

Furthermore, the metadata can be serialized to XML to allow transportation of the field information over the Internet. An example use of this is using the XML describing a page to allow an Open Archives Harvester to harvest the metadata of the pages within the collection. Section 3.3.3.2.11 will expand further on the uses of OAI to allow for metadata harvesting.

3.3.1.2.4 **Link**

In essence the link class assist in the linking of two resources in the site. A link consists of the following information:

- *Metadata*
  A limited set of metadata fields is used to describe the link. These include a unique identifier and also the type of resource to which the link points, e.g. ARTICLE.
- *Word*
  A link has an associated word, which will be the hypertext anchor that points to the resource. Typically, all occurrences of a word within a section will become a link to the specified resource.
- *URI*
  The URI is the actual location of the resource, either within the site or on the Internet. If, for instance, the link points to a local resource the URI will be a JavaScript function that passes tracking information to the module to assist in updating the user's profile.

- *Title*
  Each link can have a descriptive title that contains the title of the resource to which the link points as well as the dates and times on which the user has visited the resource. This information will assist the user's decision-making process on whether to follow the link or not.
- *Internal document ID*
  If the link points to an internal resource, the associated resource ID is required to track the user's navigational behaviour throughout the site. Also, the ID is used to load additional information that can be used to create the title of the link.

When a link is rendered in the browser, the user's personal preferences and history is taken into account to adapt the presentation accordingly. One example of such adaptation is the addition of the dates and number of times that a user has followed the current link. This information is added to the title of the link, which is displayed when the mouse hovers over the link's anchor word.

A moderator creates links between pages and resources by simply selecting a word or phrase from a page's content and then selecting the target resource. This information is then stored in the XML representing the page. Section will describe the structure of a page's XML in more detail.

3.3.1.2.5 **Section**

An page consists out of various sections, each containing its own metadata information and links. Each section contains the following properties and information:

- *Metadata*
  A reduced Dublin Core metadata set is used to describe the properties of the sections. These include the identifier, title and description. In addition to the before mentioned fields, a section also has a list of descriptive keywords and a special field that indicates whether the section can stretch or not.
- *Links*
  Each section has its own collection of links, that are instances of the Link class, which is applied to the content of the section when rendered.
- *Content*
  The content of the section can be either normal text of formatted HTML. When the content is rendered, the collection of links for the specific section is loaded and adapted based on the current user's personal preferences.

As mentioned, the links and content for each of the sections are separated. The reason for this is that it allows for better link management and link adaptation. This is known as LinkBases and has been used in, for example, Microcosm [Hall 1996], which is described in Section 2.1.2.6.

Sections contain a special field that indicates whether it has the ability to stretch. In brief, stretch text is used to hide information that is deemed to be of less importance for the user. For example, if a section contains a long non-technical introduction that is not of very high importance, a user might not want to read it again on subsequent visits. Thus, by using stretch text this particular section can be hidden and a short description can be displayed in its place. If a user wishes, he/she can then expand the full text of the section to read the entire section.

3.3.1.2.6 **Page**

An page is a collection of sections and links that can either be part of a collection or sequence of pages, or it can be a single entity. The content of the site is divided into various subjects, each containing a collection of pages. XML is used for the definition of page sections and content, which allows for user editing and also the system to parse the information. Once loaded, a page’s XML is converted to a Page class that is serialized to the server. This approach was taken to allow users to edit the XML, which takes longer to parse, and storing the serialized object, which loads much faster, for future usage by the system.

Each page also contains the full Dublin Core metadata set describing its content. In addition to metadata each section also contains a set of themes that further describes the content and allows for recommendations based on a user’s personal preferences. A weighted value is assigned to each of the page’s themes to indicate the conformance to the particular theme. For example, a page might have a theme structure similar to the following:

Data mining: 25.0

AH: 70.0

The theme information is also used to update a user’s personal preferences dynamically. Thus if a user visits a page on Adaptive Hypermedia frequently then his level of interest in this theme will be increased proportionately to the weighted value of the page and time spent reading the content.

The following sequence of events occur when a page is rendered and displayed to the user:

- *Load XML or serialized Page object*
  The XML version of the page is always considered to be the master copy. Thus, if the XML is newer than the serialized object version of the page the XML will be re-parsed and a new object will be created. This was done to allow users to edit the XML, which is both understandable to humans and computers, when the need arises, whilst ensuring that the object version of the page will be kept up to date.
- *Load metadata and themes describing page*
  The metadata information and themes describing the page is loaded from the page object and stored in an internal metadata object. This information is then used to recommend additional page that a user would benefit from reading. For example, if the current page is part of a sequence, then the collection of pages that should have been read as well as the pages that follow can be loaded and recommended to the user.
- *Load collection of sections*
  Each of the sections of the page is loaded including their associated metadata information and links. The sections are then stored in the page's internal collection of sections and rendered according to the user’s personal preferences.
- *Render content and links*
  Lastly, the content and links are rendered as HTML, or XML, for the current user. Each of the sections will be rendered according to their specific attributes. For example, a stretch text section will be rendered differently from sections marked as being of high importance. Also, each of the sections’ links will be adapted to reflect the user’s history and personal preferences.

3.3.1.2.7 **Subject**

Subjects are used to organise the collection of pages within the site. Each subject has its own separate set of themes describing the pages contained within it. Security restrictions are placed on each of the subjects to limit access to authorised users.

3.3.1.3 ***Clustering***

Pages are clustered based on their themes and associated weighted values. The reason for clustering pages is to allow for faster filtering when making recommendations to the user. Each cluster is represented by a set of themes with associated weighted values and a collection of pages. Cluster information is updated every time that a page’s theme values are changed, ensuring accurate cluster representations.

A cluster manager was developed to be responsible for the creation and maintenance of cluster information. Initially the cluster manager contains an empty collection of clusters. Once a new page is passed to the cluster manager, it will be compared to the collection of existing clusters. In the case where there are no existing clusters, a new cluster will be created around the newly added page’s theme values. All subsequent pages’ themes will then be compared to that of the existing cluster. If there are no matching clusters a new cluster will be created around the page’s theme values. This process continues until all pages have been added to at least one cluster.

In order to ensure that a page will match at least one cluster’s representation, a certain amount of leniency is used to allow pages to match loosely to a cluster’s set of theme values. For example, if there exists a cluster represented by the following themes:

Data mining: 25.0

AH: 70.0

and a new page is passed to the cluster manager with themes such as:

Data mining: 30.0

AH: 65.0

then the new page will be added to the cluster because of the leniency used when matching themes. The following pseudo algorithm describes the theme comparison used in the clustering process.

```
Get the current Page's ThemeStructure
For each of the clusters
    Get the current cluster's ThemeStructure
    For each of the themes in the cluster's ThemeStructure
        Get the intersection of the cluster's ThemeStructure and that of the page
        Increment the total difference between the ThemeStructure objects
    If the total difference is less than threshold
        Add page to cluster
```

Table 2: Pseudo algorithm for clustering process

3.3.1.4 ***Integration of modules***

One of the main goals of developing a modular framework was to allow various models to interact with one another and also with the core functionalities of the framework. For example, the content module uses the MetaDataField class defined in the Repository module and uses the interface functionalities exposed by the core Interface component.

The following section will describe the interaction between the Content module and other modules and components contained within the framework.

3.3.1.4.1 **Modules and components**

In order to reduce code redundancy, various modules can interact with one another via exposed functionalities and classes. The Content module uses the following functionalities of other modules and core components:

- *MetaDataField*
  Defined in Repository module.
  A metadata field consists of various attributes defining default values, requirements, rendering information, etc.
- *ThemeStructure*
  Defined in core Module component.
  A theme structure consists of a set of theme and weight values that are used to either describe a page's content or a user's personal preferences.
- *Various XML utilities*
  Defined in Utilities component.
  XML is used to store the page's master copy, thus various XML parsing and traversal functions are required to manipulate the XML structure.
- *Configuration settings*
  Defined in Configuration component.
  The content module has its own configuration settings that need to be accessible to the module. Also, global configuration settings need to be accessed via the root configuration component.
- *User preferences and history*
  Defined in User component.
  A user's personal preferences and history need to be updated based on the information that has just been accessed. For example, the user's personal preferences and history need to be updated to reflect the newly acquired

knowledge and to provide the user with further recommendations.

3.3.1.4.2 **Interface**

As with all the modules, the content module utilises the Interface component's rendering capabilities as well as navigational support. The interface component allows the module to present its information in different manners without the need to reconfigure the module's output. This is achieved by structuring all content and links as XML, which is then processed by the Interface and rendered using a selected XSL document.

Similarly, navigational links for the module are also presented selectively by the Interface component based on the current user action and roles. For example, a WORKER will be presented a different set of links for the management section of the content module when compared to a normal USER.

The workings of the Interface component and navigational support will be described in more detail in Section 3.4.1.

3.3.1.4.3 **Java Server Pages**

Java Server Pages are used throughout the framework to present information to the user and to handle any direct input from the user. This separation of presentation and functionality allows the developers of modules to be freed from the need to reconfigure modules for each type of interface.

Each class of the component module exposes a function that can return HTML or XML, which is used by the JSP and Interface component to present the information based on the currently selected XSL document. Similarly, links are stored in the configuration document of the content module, which is accessed by the JSP and Interface component and presented selectively based on the current user's security role and action.

The JSP is responsible for sending HTML content to the client browser, and also to handle all page requests. Based on the current user action, the JSP will instantiate the correct class of the content module and pass any required parameters, as obtained from the client's request, to the module to complete the request. The resulting HTML will then be obtained from the Interface component and presented to the client.

3.3.1.5 ***Database***

A relational database is used to store the metadata associated with each of the pages contained within the collection. The reason behind the storing of metadata in both a database and XML is to accommodate searching the collection of pages. When searching for a page, the relational database provides for far better searching criteria and speed compared to XML search functionalities. As with the updating of the page object, the database is also automatically updated every time that the master XML document is altered to ensure data validity.

The metadata schemas used to describe the pages include the full set of Dublin Core fields as well as three Learning Object Metadata(LOM), fields, viz

- *Keyword*
  The Dublin Core schema only supports full text descriptions of resources, hence the need to use the keywords field of the LOM to describe key concepts contained within a page.
- *Requirement*
  Adaptive hypermedia relies heavily on the use of user profiles which contains a history of resources that a user has accessed. By using this knowledge of the user's history and the requirements of each of the pages, the system can accurately provide the user with a collection of pages that he/she can read next. The requirement field of the LOM schema assisted in accurately describing the list of pages which are required to be read before a specific page is ready to be read by the user.
- *Difficulty*
  Not all users have the same level of understanding of all subjects, therefore some users might prefer to only read pages and sections that are on their level. The LOM difficulty field is thus used to describe the difficulty level of each page content and concepts.

In addition to the before-mentioned metadata schemas, the custom schema developed for the framework, called Malcolm, is used to define the clusters to which each page belongs.

A standard relational SQL database is used to store the metadata for each of the pages. The following fields define the structure of the database:

Field name	*Field type*	*Field description*
dc_title	varchar(100)	The title of the page.
dc_creator	varchar(100)	The creator of the page's content.
dc_subject	varchar(100)	The subject of the page.
dc_description	blob	A brief description of the page's content.
dc_publisher	varchar(100)	The publisher, if any, of the page.
dc_contributor	varchar(100)	An optional contributor to the content of the page.
dc_date	datetime	The date on which the page was created.
dc_type	varchar(32)	The type of resource that is described.
dc_format	varchar(16)	The format of the page's content. Default is text/xml.
dc_identifier	varchar(36)	A unique identifier assigned to the page.
dc_source	blob	The original source, if any, of the page.
dc_language	varchar(5)	The language of the page's content, e.g. en-GB.
dc_coverage	blob	The domain area which the resources covers.
dc_relation	blob	The relationship to other resources.
dc_rights	blob	Any associated copyright information.
Lom_requirement	varchar(255)	A list of required page ID's for this page.
Lom_difficulty	varchar(10)	A weighted value indicating the difficulty of the content.
Lom_keyword	blob	A list of keywords describing the page.
malcolm_clusters	varchar(255)	A list of cluster ID's to which this page belongs

Table 3: Field definitions for the Content's XML storage

XML is used as the primary master storage medium for page information. The reasoning behind the use of XML is because of its ability to accommodate complex data structures and also its human and computer readability. Primarily a page's XML consists of descriptive metadata and a collection of sections. Appendix A contains an example XML document representing a page.

3.3.1.6 ***Adaptation***

Adaptation of content has to do with using a user's personal preferences and history. A user model is used to represent the user's preferences and assumed knowledge of the domain. There exist two main adaptation techniques, as defined by [Brusilovksy 1998] [Bailey 2002], viz. adaptive presentation and adaptive navigation. This section focusses on how adaptation has been implemented in the framework. For a more detailed information such as the reasoning for adaptation and the strategies that can be used, refer to Sections 2.1.2.1 and 2.1.2.5.

3.3.1.6.1 **Adaptive presentation**

Adaptive presentation is concerned with customizing the layout and presentation of content to the user based on his personal preferences. Methods used by the content module include stretch text that will conditionally display certain sections if a user has read them before and the content moderator has deemed them to be of less importance.

For example, if a section contains an introduction to the page and is marked as being stretchable, then once the user has read the page and returns on a subsequent visit the section will be hidden and a brief description will be displayed in its place. The user can then expand the full contents of the section if he/she wishes. Another use of stretch text is to hide information that a user might not find interesting or that has another difficulty level than that of the current user.

Adaptive presentation can thus be used to reduce page clutter as well as to assist a user in reading only relevant information.

3.3.1.6.2 **Adaptive navigation**

Adaptive navigation allows for the customisation of link presentation and ordering. The main method of adaptive navigation used was link adaptation. Link adaptation is the process of adapting the presentation of links and the addition of useful information. For example, the user's personal history is used to determine the dates and number of times that the user has followed the link. Also, by using the end point of the link, information regarding the destination of the link is be added, such as the title of the page and difficulty level of the content. Another example of the use of adaptive presentation is the altering of link visibility by using various colours to denote the perceived importance of the link.

A crucial component of adaptive navigation is the user's personal preferences that contain information like a user's interests and preferred style. Also, the user's history is important to determine the dates on which the link was accessed as well as whether the link is of high importance. It is assumed that resources visited often are deemed to be important to the user.

Further methods of adaptive navigation that can be implemented in the future is adaptive ordering, where links are ordered based on their perceived importance to the current user. This method will also make extensive use of the user's personal preferences and history.

3.3.1.7 ***Recommendation system***

In order to assist the user in finding relevant information, recommendations are made based on his personal preferences and history. A user's history is used to determine relevant pages based on other pages' requirements, and a user's personal interests are compared to that of pages contained within the collection.

3.3.1.7.1 **Requirements and history**

A user’s navigational behaviour is tracked throughout the site in order to create a history containing the list of pages that a user has read. Combing a user’s history and requirements of sequential pages the system can recommend pages that might be of interest to the user.

For example, the collection contains pages A, B and C, with page A being a requirement for B and C. Assuming the user is reading page A, then he/she will be presented with a list of recommended further reading containing both pages B and C. If the user moves on to page B, he/she will be presented with a list of recommended pages, that he/she should have read by now, containing page A.

These types of recommendations are useful when dealing with instructional material where one page logically follows another.

3.3.1.7.2 **Personal preferences**

A user’s personal preferences consist of a collection of interests each with a weighted value. This allows for a structured representation of a user’s interests, which can be used when recommending additional pages.

Recommendations are made by comparing a user’s own personal preferences with that of the collection of page clusters. Each cluster of pages is represented by a set of themes each with a weighted value that is used to determine the relevance to the current user’s interests. In order to assure that recommendations are made without a user’s personal preferences having to match exactly to one of the clusters, a certain amount of leniency is used. This leniency thus allows for fuzzy matching of user and page clusters.

The following pseudo algorithm describes the process of matching a user's preferences with the ThemeStructures of the pages when making recommendations.

```
Get the current Users's ThemeStructure
For each of the pages
    Get the current pages's ThemeStructure
    For each of the themes in the pages's ThemeStructure
        Get the intersection of the page's ThemeStructure and that of the user
        Increment the total difference between the ThemeStructure objects
    If the total difference is less than threshold
        Add page to list of recommendations
```

Table 4: Pseudo algorithm for recommendation process

Using the set of user clusters, the system can make another set of recommendations. User recommendations are made by comparing the history of users, in a user's cluster, with that of the current user. The set of recommendations consist of two main types, viz.

- *Users*
  All users, with public profiles, within the current user's clusters are displayed in a list, allowing the user to contact them if he/she wishes.
- *Pages*
  The histories of similar users are compared and then using the pages appearing most frequently a collection of recommended pages is then created. This list is then presented to the user as recommended reading.

3.3.1.8 ***Content Management***

An important aspect of the content module is the ability to administer the content contained within the collection. The content management system allows for the following administrative functions, viz. submission and removal of content, editing of themes and requirements of pages and link management.

3.3.1.8.1 **Content submission**

Each page in the collection is described by its own set of Dublin Core metadata. By using easy to use form fill-ins and a set of required steps, the content module's content submission function ensures the validity of the XML file.

On submitting a new page, the required metadata fields need to be filled in and is validated using a set of Validators. Once the metadata has been supplied, each of the sections of the page needs to be created. Each content section has a reduced set of metadata consisting of a title, description, a set of keywords and whether the section is stretchable. The content module is then responsible for the creation of the page from the XML file as well as the serialized object version of the page. Also, the metadata describing the page is added to the page database.

3.3.1.8.2 **Theme management**

Themes are used to describe the content of each of the pages contained within the collection. Each of the pages contains a set of themes each with a weighted value.

Administrators can edit the themes describing any of the pages in the collection via the content management interface. The procedure consists of selecting the desired page and then simply selecting a weighted value for each of the themes to describe the content of the page. Once the page's themes have been updated the cluster manager will re-cluster the current page to ensure the validity of clusters.

3.3.1.8.3 **Link management**

Links in the content of a page are stored separately from the content sections to allow for easy maintenance and adaptation. Each link is described in the XML as a separate node containing descriptive metadata and linking information.

The following steps are required in order to add a new link for a specific page:

1. *Choose a section to contain the link*
   Once a page has been selected the user is presented with a collection of the page's sections. Each section can contain any number of links, each specific to the current section.
2. *Selecting a word or phrase*
   Links are basically anchors on specific words or phrases within the page's sections' content. On selecting a word or phrase, the next step is to supply required link information.
3. *Fill in required link information*
   Each link must define the type of resource to which the link points, the unique identifier of the resource and a title describing the link. This information is gathered using the link assistant that requires the user to complete a set of steps regarding the link properties.

Once all the link information is gathered, the new link is created and the page's XML file and associated serialized object document are updated to reflect the changes.

3.3.1.8.4 ***Page requirements***

Page requirements are used for recommendations and sequencing of pages within the collection. Each of the pages in the collection can have a list of required pages that need to be read prior to the current one.

Requirements are made using the management interface by simply selecting a page for which to edit the requirements, and then selecting each of the required pages from the collection. Once all the required pages have been selected the XML file and serialized object representing the page are updated as well as the page database.

3.3.1.9 ***Configuration***

All relevant configuration information for the content module is stored in its associated XML configuration file. The configuration manager is responsible for loading the initial configuration values and then monitoring the configuration file to ensure that when changes are made, they are automatically loaded to ensure validity of configuration information. See Appendix A for an example configuration file for the content module.

3.3.1.9.1 **Standard settings**

Standard settings are settings required by the framework to allow for successful integration with the core components and other modules. The standard settings for the content module is as follows:

- *Events & Plugins*

  Events are triggered by various actions within the module, for instance the addition of a new page. For each of the events there is a collection of associated plugins that are executed to allow for customisation of the module.

- *Navigation*

  The navigation sections contain a collection of links used by the JSP of the module. Each link consists of a type, URI, title, required condition and optional JavaScript code.

- *Help*

  Context sensitive help information is defined here, which is used by the JSP and module to present help on various actions.

3.3.1.9.2 **Additional settings**

In addition to the required standard settings, the Content module also relies on additional settings to be supplied. Additional settings are module specific, and consist of the following:

- *Database*
  The database settings contain information on the creation of the SQL database and also any default values to be inserted into the database. Basically, this section defines a set of SQL statements that can be used by an administrator when installing a new instance of the framework.
- *Subjects*
  The content of the site is divided into a set of subjects, which is defined here in the configuration document. Each subject entry consists of an ID, name, description and default user role. Also, each of the themes representing the subject is listed with a weighted value for each. Lastly, the security constraints are defined for each of the associated actions, for example the required role necessary to submit a new page.
- *Metadata*
  Metadata is used to describe the properties of each of the pages contained within the collection. This section defines the set of metadata fields used to describe the pages.
- *Validators*
  Validators are used to ensure the validity of information supplied by the user when, for instance, a new page is submitted. This section contains a collection of Validators each representing an HTML form field. The Validators are grouped together to allow for selective use, based on the current action. For example, there exists a group of Validators associated with submitting a new page.

3.3.2 Discussion forum

3.3.2.1 ***Overview***

The online discussion forum allows for a fully threaded topic-based forum. Users can select from any of the available topics, post messages or reply to existing messages. Each topic has its own security constraints to limit access to authorized users. In addition to limiting access to certain users, the security constraints can also limit actions available to users based on their security roles.

Forum topics are described by a title, description and also a set of themes. As with the content module’s pages, each topic is described by a set of themes each with a weighted value. Topics can be recommended to users by comparing a user’s personal interest with the weighted values of the forum.

*3.3.2.2 **Classes***

The discussion forum module comprises of the following collection of Java classes.

3.3.2.2.1 **Module**

The module class is found throughout the framework, and is essentially a specialization of the root Module class that defines and exposes certain required functionalities. Most of the required functionalities have already been defined in the root Module class, thus requiring very little specialization of the class.

An example of specializing the module is the loading of security roles associated with each action. The availability of actions to users is based on security constraints, for example only administrators can edit the existing messages in the forum.

3.3.2.2.2 **Loader**

The loader is also a specialization of the root Loader class and is primarily concerned with loading initial configuration values, such as security roles, from the configuration files. A thread runs in the background to assure that any changes made to the module’s associated configuration information is picked up and handled accordingly.

3.3.2.2.3 **DiscussionMessage**

Each message in the discussion forum is represented by a DiscussionMessage instance. A database is used to store message information and a new DiscussionMessage object is created when a message is to be viewed. The following properties define a message in the discussion forum:

- *Identifier*
  A unique identifier is assigned to each message within the discussion forum. Identifiers typically comprise of the message’s parent topic and a unique numerical value.
- *Topic*
  The identifier of the topic containing the message is used in order to locate and correctly represent messages in the discussion forum.
- *Type*
  A message can either be a new post or a reply to an existing message.
- *Parent*

If the message is a reply to an existing message, the parent will be the identifier of that message. Otherwise, the parent will be ROOT to indicate that the message is a topic level post within the topic.

- *Level*
  The level of the message is the depth of the message in the particular topic. For instance, a new post will be level 0 and a reply to a message will be one level deeper than the parent message. The level of the message will determine the indentation of the message in the threaded display of the messages within the topic.

- *Author*
  The author field of a message contains the username of the user who submitted the message and is automatically set as the default value when a user posts a message to the forum.

- *Date*
  This is the date and time on which the message was submitted to the discussion forum.

- *Subject*
  Each message can have a unique subject or continue with an exiting message subject. Replies to messages typically have "RE:" appended to the start of the parent messages' subject to indicate that the message continues a previous discussion.

- *Body*
  The body of the message is the actual content of the message. By default, all HTML is stripped from the message body, but this can be changed in the configuration settings.

3.3.2.2.4 **DiscussionTopic**

The discussion forum consists of various topics that are used to logically group the collection of messages. A topic consists of a name, description and a collection of themes describing the topic. Also, each topic defines its own security constraints to limit access and actions to authorized users. Very few actions are defined in the DiscussionTopic class, which is primarily used as an abstraction to assist in the grouping of messages.

3.3.2.2.5 **Discussion**

The Discussion class performs most of the functions associated with the discussion forum. These functions include posting messages, querying the collection of messages and loading topics from the XML configuration file.

3.3.2.3 ***Integration***

One of the main goals of developing a modular framework was to allow various models to interact with one another and also with the core functionalities of the framework. For example, the discussion forum module uses the interface functionalities exposed by the core Interface component.

The interaction between the discussion forum module and other modules and components contained within the framework are similar to that of the content module as described in Section 3.3.1.

3.3.2.4 ***Database***

A relational database is used to store message information for each message within the discussion forum. SQL is used to query the database for specific message information, for instance when creating a new DiscussionMessage object. A single database table is used to store all the messages for each of the topics.

Field name	*Field type*	*Field description*
id	int(11)	A unique identifier for the message.
topic	varchar(50)	The name of the topic to which the message belongs.
type	enum('post','reply')	The type of message, either post or reply.
parent	varchar(16)	The identifier of the message's parent message, or ROOT if the message is a new post.
level	smallint(6)	The level of the message in the thread.
author	varchar(50)	The username of the author.
date	timestamp(14)	The date and time on which the message was submitted.
subject	varchar(255)	The subject of the message.
body	blob	The body content of the message.

Table 2.5. Field definitions for the Discussion Forum table.

3.3.2.5 ***XML & XSL***

The threading of messages in the discussion forum requires a recursive function to properly build the tree of messages. An XML tree is constructed representing the threading of the messages for a certain topic. Once the tree has been constructed a recursive XSL style sheet is applied to the XML to transform the tree into HTML.

3.3.2.6 ***Functionalities***

The discussion forum allows for the following actions to be performed depending on their security levels:

- *View topics*
  A list of all the topics comprising of the title, description, number of posts, replies and active users is presented to the user. Each topic has its own set of security constraints that will limit certain actions to authorized users.
- *View messages for a topic*
  Once a topic has been selected, the user is presented with the threaded display of the messages contained within the forum. Messages are indented based on their level in the tree to indicate new posts and replies. Also, the themes describing the current topic are displayed to provide further information on the type of messages that belong to the topic.
- *View a specific message*
  A user can view the detailed contents of a message by selecting the subject from the threaded list. The message details consist of a subject, author, date and message body. Also, a user can view the list of replies to the current messages with links to each message’s details.
- *Reply to an existing message*
  When viewing a message, a user can also reply to the current message. The reply will then be inserted into the database with the appropriate information to indicate the parent and level of the message.
- *Post a new message*
  Posting a new message allows a user to start a new discussion or ask a question pertaining to the selected topic. The only required information for a new message is the topic, subject and content of the message. Default values are automatically inserted for the author and date of submission.
- *Search for messages*
  A user can search for messages in the discussion forum by supplying a word or phrase and the desired topic. Search results consist of the message’s subject, author and date of submission, with the subject linking to the detailed message view.

3.3.2.7 ***Forum management***

A moderator can administer messages within the forum via the web based management page. The content of a message can be altered if the need arises or the message can be completely removed from the discussion forum.

Forum topics can only be managed by altering the discussion forum's associated XML configuration document.

3.3.2.8 ***Configuration***

All relevant configuration information for the discussion forum module is stored in its associated XML configuration file. The configuration manager is responsible for loading the initial configuration values and then monitoring the configuration file to ensure that when changes are made they are automatically loaded to ensure validity of configuration information. See Appendix A for an example configuration file for the discussion forum module.

3.3.2.8.1 **Standard settings**

Standard settings are settings required by the framework to allow for successful integration with the core components and other modules. The standard settings for the discussion forum module is as follows:

- *Events & Plugins*
  Events are triggered by various actions within the module, for instance the addition of a new document. For each of the events there is a collection of associated plugins that are executed to allow for customisation of the module.
- *Navigation*
  The navigation sections contain a collection of links used by the JSP of the module. Each link consists of a type, URI, title, required condition and optional JavaScript code.
- *Help*
  Context sensitive help information is defined here, which is used by the JSP and module to present help on various actions.

3.3.2.8.2 **Additional settings**

In addition to the required standard settings, the module also relies on additional settings to be supplied. Additional settings are module specific, and consists of the following:

- *Database*

  The database settings contain information on the creation of the SQL database and also any default values to be inserted into the database. Basically, this section defines a set of SQL statements that can be used by an administrator when installing a new instance of the framework.

- *Topics*

  Discussion forum topics are each represented by an XML node comprising of an identifier, name, description and collection of themes. Each topic also defines a set of security constraints for each action associated with the discussion forum.

- *Validators*

  Validators are used to ensure the validity of information supplied by the user when, for instance, a new message is submitted. This section contains a collection of Validators each representing an HTML form field. The Validators are grouped together to allow for selective use, based on the current action. For example, there exists a group of Validators associated with submitting a new message.

3.3.3 Repository and workspace

3.3.3.1 ***Overview***

In essence the repository is a hierarchical storage of documents of any format. Documents can be retrieved by either browsing or using the search functionality. The repository module allows for the creation of a fully functional digital library supporting versioning, hierarchies, browsing, searching, etc.

Documents are submitted to the repository, accompanied by descriptive metadata, and stored in the appropriate location in the hierarchy. A relational database is used to store the metadata to allow for fast searching. Also, on submission of a document a reverse index of the content is created to allow for full text searching of documents.

The Repository Module was developed to be flexible enough to allow for implementation in the following two formats:

1. *Digital Library*

   When implemented as a digital library, the repository limits the functionalities that can be performed on the documents contained within. In essence, a digital library is used to store documents as they appear in their final format. It is therefore suited for documents or articles that have been through a peer review process which serve to provide users with valuable reference material.

2. *Workspace*

   Unlike a digital library, a workspace allows users the ability to change documents contained within the repository. Users can check-out a document to make changes and then submit a new version of the document to the repository. It can therefore be used in a collaborative environment where documents are not yet finalised and need to evolve in conjunction with the project.

3.3.3.2 ***Classes***

In order to develop the content module the following set of classes were created.

3.3.3.2.1 **Module**

The Module class is found throughout the framework, and is essentially a specialization of the root Module class that defines and exposes certain required functionalities. Most of the required functionalities have already been defined in the root Module class, thus requiring very little specialization of the class.

An example of specializing the module is the loading of security roles associated with each action. The availability of actions to users is based on security constraints, for example only administrators can edit the hierarchy of the repository.

3.3.3.2.2 **Loader**

The loader is also a specialization of the root Loader class and is primarily concerned with loading initial configuration values, such as security roles, from the configuration files. A thread runs in the background to assure that any changes made to the module's associated configuration information is picked up and handled accordingly.

3.3.3.2.3 **MetadataField**

Metadata is used extensively throughout the repository to describe document properties to allow for proper and structured storage and retrieval. A metadata field is used to describe a specific property, for instance the title, of a document. The metadata fields describing the document is defined in the configuration XML and consists of the following components:

- *Name*

  Each field is identified by a unique name, for instance title. The field name is used to store and retrieve the field via the metadata class.

- *Description*
  A brief description of the field, which can be used to assist a user when submitting a document.
- *Type*
  The type of field can either be TEXTBOX, LIST or TEXT. Each field will be rendered differently depending on its type.
- *Label*
  A label is presented next to the field when, for instance, a user submits a new document. Labels are typically not more than two words.
- *SQL field name*
  This is the field's associated field name within the repository database.
- *SQL create statement*
  Each field defines its own SQL statement that is used to create the field in the repository database. This will assist administrators when creating a new instance of the repository or framework.
- *XML Namespace*
  XML namespaces are used to eliminate ambiguity between fields with the same name.
- *Required*
  A Boolean value indicating whether the current field is a required field or not.
- *Order by*
  A Boolean value indicating whether this field can be used to order search results.
- *Primary*
  A Boolean value indicating whether this field is one of the primary fields. Primary fields are those that are displayed to the user when submitting a new document.
- *Collection of values*
  Each field can have a default value as well as a list of values from which the user must select at least one. Restricting a user to a certain collection of values can ensure data validity and correctness.
- *Validator*
  Validators are used to ensure that users fill in required fields and that field values are valid. For example, a Validator can be used to ensure that a date field is in the format yyyy-mm-dd. Validators will be discussed in further detail in Section

3.4.1.2.2.

Each metadata field is rendered based on its type and current values. The different types of fields are:

- *Text box*
  Text boxes are single line input fields that are used primarily for properties with shorter values. An example of a text box is the title field of a document. Values can be constrained using Validators to ensure data validity, for example date formats.
- *Text*
  Text fields are used for fields requiring more descriptive information, for example the description of a document. As with text boxes, the values of text fields can be constrained using Validators.
- *List*
  List fields are used when a user has to select from a list of values. An example use is the subject field of a document. An advantage of using lists is that users are forced to use a value that is recognized by the repository, which will allow for better storing and retrieval of documents. List values are defined in the XML configuration document describing each of the metadata fields.

3.3.3.2.4 **Metadata**

Metadata is used to describe the properties and attributes of a document. In essence, the metadata class consists of a collection of fields that can be used to describe a resource. The field information is loaded for the module's configuration file and stored in a hash table to allow for lookups.

In addition to containing a list of all the fields, the metadata class also assists in ensuring that required information is supplied when submitting a new document by utilizing the Validator component.

Furthermore, the metadata can be serialized to XML to allow transportation of the field information over the Internet. An example use of this is using the XML describing a document to allow an Open Archives Harvester to harvest the metadata of the document within the repository. Section 3.3.3.2.11 will expand further on the uses of OAI to allow for metadata harvesting.

3.3.3.2.5 **File**

A file represents a document stored on the server somewhere in the repository's hierarchy. Primarily, a file is used by the Explorer class to construct the XML representation of the repository hierarchy. A file contains only the information necessary to construct the XML hierarchy. This information exists of the document's identifier, parent folder in the hierarchy, title, creator, date and whether the document has been accepted or not. The hierarchy will then be constructed using the XML representing the file.

3.3.3.2.6 **Hierarchy Explorer**

The hierarchy explorer performs a recursive traversal of the directories contained within the hierarchy to build an XML representation of the hierarchy. Once the XML representation has been created it can either be rendered using XSL or used for any other purpose. One such purpose is to allow an OAI Harvester to determine the collection of sets contained within the repository. For more information on OAI Harvesting, see Section 3.3.3.2.11.

Depending on the current required action of a user, the hierarchy explorer can either only construct the directories defining the hierarchy, or the list of files contained within each directory can also be loaded. In the case where file information is also required, the explorer will make use of the File class to define the properties of each file contained within a directory.

Finally, for presentation an XSL document can be used to recursively traverse the hierarchy of the repository to present the repository's directories and files to the user. By using XSL for the presentation, one can easily switch between presentation styles without the need to recompile the module or alter the XML representation.

3.3.3.2.7 **Directory**

A directory represents a logical node of the hierarchy representing the repository. A database is used to describe the collection of directories in the hierarchy. Also, each directory is represented by a folder on the server that will contain the actual documents contained within the repository. The information describing a directory consists of:

- *Identifier*
  A unique identifier used to access the directory. Each identifier is derived from a directory's parent directory's identifier.
- *Path*

The path to the folder, on the server, representing the directory in the hierarchy. This path's structure will match that of the hierarchy.

- *Parent*
  The identifier of a specific directory's parent directory. This is used for reverse lookups when traversing back up the hierarchy.
- *Level*
  An integer value denoting the level of a specific directory in the hierarchy. The root of the hierarchy is level 0 and each subsequent sub-folder is one level higher.
- *Title*
  A title describing the directory that is used when presenting the hierarchy.
- *Role*
  A required security role that a user must have to access the current directory. This security level will cascade down to every sub-directory in the hierarchy.

Each directory also contains a list of File objects that each represents a document contained within the directory. All of the information describing a directory, such as properties and files, are converted to XML, which is then rendered using XSL.

3.3.3.2.8 **DocumentSearch**

The repository is fully search able allowing for simple, advanced and full text searches. All search related operations are defined in the DocumentSearch class. The DocumentSearch class will perform a search for documents using the supplied criteria and render the search results using HTML. Searching is made possible and effective due to the use of metadata to describe document properties and the subsequent storing of this information in an SQL database.

For more information on the repository's search functionalities, see Section 3.3.3.6.

3.3.3.2.9 **Document**

The Document class is the core of the repository and represents the documents contained within the hierarchy. A document consists primarily of metadata describing its properties and the actual document content, for instance an Adobe Acrobat Document. Once a user requests a document a new instance of the Document class is instantiated and the appropriate properties of the requested document is loaded.

The following steps are taken when a user requests a specific document in the repository:

1. *Extract repository information*
   A document's identifier comprises of the identifier of the repository and a unique number of the document. It is necessary to thus extract the identifier of the repository to be able to load the correct instance, which contains useful information for the presentation of the document.
2. *Load properties*
   An additional property, which is not loaded by the metadata, is the collection of users that need to be informed once a document has been unlocked. The list of usernames is loaded and stored internally to allow for notifications.
3. *Load metadata*
   Each document is described using a set of metadata fields stored in a relational database. This information is thus loaded from the database via the document's metadata class and stored internally.
4. *Load document versions*
   Various versions of the same document can exists side-by-side, thus each version is described by its own metadata structure and has an entry in the repository database. It is thus necessary to load a reduced set of metadata describing each version of the document to allow the user to access any of these versions.
5. *Render document information*
   Lastly, once all the information has been loaded a document has to be rendered using HTML. The HTML formatting is defined in the Document class and also the repository module's XML configuration document. There exist two main tasks when rendering the document, viz. presentation of metadata and the presentation of the actual document content. More information on the rendering of the document and available functionalities will be discussed in Section 3.3.3.5.

3.3.3.2.10 **RepositoryInstance**

Various repositories can co-exist within the repository, each being an instance of the RepositoryInstance class. Each instance is in essence only an abstract class that contains some information describing the repository as well as a set of functions commonly used throughout the repository. These functionalities include the loading of metadata fields from the XML configuration document, accessing certain subsets of the metadata fields and information describing the repository.

Each document is aware of its parent repository's identifier and is thus able to load the correct instance and access specific functionalities. An example of where a document uses its parent repository is when determining what information is to be rendered as HTML, such as metadata.

3.3.3.2.11 **Open Archives Initiative (OAI)**

The repository is Open Archives compliant and uses the functionalities exposed by the OAI class to allow for metadata harvesting. These functionalities include the loading of information, serialization of document metadata to XML, etc. OAI functionalities will be discussed in further detail in Section 3.3.3.2.11.

3.3.3.3 ***Integration***

One of the main goals of developing a modular framework was to allow various modules to interact with one another and also with the core functionalities of the framework. For example, the repository module uses the interface functionalities exposed by the core Interface component.

The interaction between the repository module and other modules and components contained within the framework are similar to that of the content module as described in Section 3.3.1.

3.3.3.4 ***Hierarchy & Document storage***

The repository is based on a hierarchical storage that represents the logical structure and ordering of the documents contained within. The hierarchy consists of a root directory with a collection of sub-directories, each containing its own set of sub-directories and documents. Documents are submitted to the appropriate directory within the repository to allow for logical browsing and searching.

The hierarchy of the repository is defined in a database as well as represented by folders on the server. A database was used to allow for easier maintenance and to enable for a directory to have a title that is not restricted to the identifier. Also, more information can be stored in a database regarding the properties of a directory, such as its identifier, title, level and required security role.

Security constraints are placed on each of the directories in the hierarchy to restrict access to authorized users. This allows moderators to create, for instance, directories for each project within a corporation and only allowing access to project members. A directory’s security role cascades down to all sub-directories and only higher roles can be applied to these directories.

The repository hierarchy is presented to the user as a tree structure that contains the titles of sub-directories and the collection of documents within the current directory. A user can then either choose to view a directory's sub-directories or select a document from the collection. Each document is represented by its title, date of creation and author.

As mentioned earlier, the metadata describing a document and the actual content of the document are stored separately. The descriptive metadata information is stored in a relational database, whereas the document's actual content, for example Adobe Acrobat Document, is stored on the server in the appropriate folder.

*3.3.3.5 **Presentation & Functionalities***

An important aspect of the repository is the presentation of a document and its associated metadata. Much attention must be paid to this because of the importance of allowing a user to easily understand the information and functionalities on the page. The presentation of a document is divided into three main sections, viz. the metadata, actual document content and a collection of available functionalities.

3.3.3.5.1 **Metadata**

The metadata describing the document is divided into three main sections, viz.:

- *Bibliographic details*

  The default collection of metadata fields describing the bibliographic information regarding a document consists of the title, author, subject, date of creation and status.

- *Document abstract*

  The document abstract is defined in the Dublin Core description field of the document's metadata

- *Full metadata information*

  Here the full collection of metadata fields describing the document is listed. This contains the full Dublin Core metadata set as well as additional information used by the framework.

The default presentation of a document's metadata is limited to the bibliographic details; however, the user can choose to view the additional metadata information.

3.3.3.5.2 **Document content**

The document content is the actual content of the original document or information object and can be stored in various formats, such as Adobe Acrobat Documents, JPEG images or AVI video's. By default, the content is not displayed and the user is presented with the option to view the content in the current browser or open the document in a new browser window. Each of these available functions are defined in the associated XML configuration and can be limited to certain repository instances. For example, the workspace repository allows for the additional function of checking-out the document for editing, which effectively locks the document until the user has resubmitted a new version.

3.3.3.5.3 **Available functionalities**

Various functions are available to the user depending on the repository instance and the security level of the current user. The functionalities for each instance of a repository are also determined by the type of implementation, viz. digital library or workspace. The functions available consist of the following:

- *View document metadata*
  This allows the user to either view or hide the bibliographic details, full document abstract or full metadata information for the current document. This function is available for both digital libraries and workspaces.
- *View location of document in the hierarchy*
  The location of the current document in the repository hierarchy is displayed, and each of the parent directories can be browsed. This function is available for both digital libraries and workspaces.
- *Available document versions*
  Displays a list of all available versions of the current document, with links to each version. This function is only available for workspaces.
- *OAI representation of document's metadata*
  Documents contained within the repository can be represented as and XML document that is OAI compliant. This function is available for both digital libraries and workspaces.
- *Edit the document's metadata*
  Allows the user to edit any of the metadata fields describing the current document. This function is available for both digital libraries and workspaces.
- *Revise the current document*
  A user can post a revised version of the current document to the repository. This function is limited to workspaces.

- *Unlock the current document*
  Allows a user to unlock the current document if it has been checked-out by another user for editing. This function is only available in workspaces.
- *Remove the current document*
  The current document can be removed from the repository using this function. This function is available for both digital libraries and workspaces.

Each of the available functions are limited to authorized users. The security constraints for each function are defined in the repository module's associated XML configuration document and can be customized for each repository instance.

3.3.3.6 ***Search***

The search functionality allows users to search the collection of documents using a set of criteria. There are three types of search interfaces available to the user, viz. simple, advanced and full text.

Search results presented to the user are grouped according to the document's location in the hierarchy. Each search result contains the title, author and date of creation for the document, with the title of the document being a link to the full document. To assist a user in choosing a document, its abstract is displayed in a small pop-up window when the mouse is pointed to the document's title.

3.3.3.6.1 **Search interfaces**

3.3.3.6.1.1 **Simple**

The simple search is the most basic of the three interfaces and provides for only limited accuracy. A user can search for a single word or phrase occurring within one or more of the available metadata fields. Search results thus consist of documents where at least one of the selected metadata fields contains the search word or phrase. For example, a user can search for documents containing the word 'Hypermedia' in either the title or abstract of the document.

3.3.3.6.1.2 **Advanced**

The advanced search allows for more accurate searching because of its ability to limit search words to specific fields. In other orders, a user can search for documents containing the word 'Hypermedia' in its title and that were created by a person with the surname 'Jones'. The collection of available metadata fields comprises of the title, subject, location in repository, author's first and last name, and date of the document.

3.3.3.6.1.3 **Full text**

In addition to searching the metadata describing documents, a user can also search the full content of a document using a full text search engine. A reverse index is created for each of the documents contained within the repository. The global index is updated once a new document has been submitted. A search algorithm has been developed to allow for the searching of a single word or a collection of words that must appear in the correct order.

```
The search phrase is split into an array containing single search words.
Remove any non-word characters from each of the words.
Remove any stop words that are commonly found in the English language.
For each of the search words do the following:
Select the documents containing the search word and the position of the word.
Update the list of documents with the occurrence of the word and position.
For each of the documents in the search results do the following:
Determine whether words in document are in correct order based on search phrase.
Add result to list only if correct number of words match and ordering is correct.
Sort search results alphabetically using each document’s title.
```

Table 6: Pseudo description of the full text search algorithm.

3.3.3.7 **Database**

An SQL database is used to store the metadata describing the document in the repository as well as the hierarchy of the repository. All search queries are presented to the database using standard SQL syntax using the JDBC database driver. The two main repository tables in the database are for the documents and directories of the hierarchy.

Field name	*Field type*	*Field description*
ID	varchar(36)	A unique identifier of the directory.
path	varchar(200)	The path to the folder on the server.
parent	varchar(36)	The identifier of the directory’s parent.
level	char(3)	The level of the directory in the hierarchy.
title	varchar(200)	A title describing the directory.
role	int(11)	The required user role to access the directory.

Table 3.7. Field definitions of the Directory table.

The following table is used to describe the documents contained within a repository. It uses a combination of the Dublin Core metadata schema as well as a custom schema, called Malcolm, that was developed specifically for the framework. The reasoning for the development of a custom metadata schema was because specialisation of the framework's Repository Module and the inability of the existing schemas in those areas.

Field name	*Field type*	*Field description*
malcolm_parent	varchar(36)	The identifier of the parent directory.
malcolm_path	varchar(255)	The path to the document on the server.
malcolm_submittedBy	varchar(16)	Username of the person who submitted the document.
malcolm_locked	tinyint(1)	A Boolean indicating whether the document is locked.
malcolm_notifyList	char(1)	A list of usernames of users to be notified once the document becomes unlocked.
malcolm_accepted	tinyint(1)	A Boolean indicating whether the document has been accepted.
malcolm_version	varchar(16)	The version of the document, e.g. 1.1.
malcolm_status	varchar(16)	The status of the current document, e.g. draft.
malcolm_visibility	enum	Either private or public visibility.
malcolm_instance	varchar(16)	The name of the framework instance containing the document.
malcolm_role	int(11)	The role required to access the document.
dc_title	varchar(255)	The title of the document.
dc_creator	varchar(100)	The author of the document.
malcolm_creatorContact	blob	Contact details of the document’s author.
dc_subject	varchar(100)	The subject of the document.
dc_description	blob	A description of the document.
malcolm_keywords	varchar(255)	Keywords describing the document.
dc_publisher	varchar(100)	The publisher of the document.
dc_contributor	varchar(100)	Names of contributors to the document.
dc_date	date	The date on which the document was published.
dc_type	varchar(32)	The type of document, e.g. Conference paper.
dc_format	varchar(36)	The format of the document, e.g. text/html.
dc_identifier	varchar(36)	A unique identifier for the document.
dc_source	blob	The original source of the document.
dc_language	varchar(5)	The language in which the document is written, e.g. en-GB.
dc_coverage	blob	The domain area which the resource covers.
dc_relation	blob	The relation of the resource to other resources.
dc_rights	blob	Any copyright information regarding the document.

Table 3.8. Field definitions of the Documents table.

3.3.3.8 ***Document management***

3.3.3.8.1 **Document submission**

Submitting a new document to the repository comprises of the following three steps:

1. *Fill in metadata fields*

   The first step in submitting a new document is the supplying of all required metadata information. A user is presented with a collection of fields, which must be completed when submitting a new document. Depending on the type of information required, a user is either presented with text boxes used for free form text, or drop down lists, used in the event that specific pre-defined values need to be selected. All supplied information will be validated using the appropriate group of Validators as defined in the XML configuration document.

2. *Select location in hierarchy*

   Once the metadata has been supplied, the user will be presented with the hierarchy of the repository and he/she must then select the appropriate directory for the document. The location in the repository will depend on the document's requirements, for example the document could be a specific department's research report and must be placed in that department's research report directory.

3. *Upload document content*

   The last step in submitting a new document is the actual uploading of the document's content, which can be in any format, for example Adobe Acrobat Document. Once the document has been uploaded it will be stored in the appropriate folder on the server with a file name matching the identifier of the document, e.g. Library123.document.

3.3.3.8.2 **Editing, revision, accepting and removal of documents**

A requirement of any digital library is the ability to manage the existing collection of documents in the repository. The repository module allows for the following administrative functions for existing documents:

- *Editing existing documents' metadata*

  A moderator has the ability to edit the metadata of a document after it has been submitted to the repository. All of the metadata fields are editable, except for the identifier of the document, which needs to remain consistent to ensure data integrity, which is read only. Any modifications to the document's metadata is validated in the same way as when a new document is submitted to the repository.

- *Submitting revised versions of documents*
  The repository allows for various versions of the same document to exist side by side in the digital library. Once a document has been submitted, a user can check-out the document for editing and then submit a revised version of the document. Once a document has been checked-out, the document is locked until the user submits a revised version, ensuring that version control is enforced. Each version of a document inherits the metadata of the original version, whilst allowing for editing of any fields' value.
- *Accepting new documents*
  A new document that is submitted to the repository is not automatically accepted, and will thus not be visible to users. In order to ensure content quality a document must first be reviewed by a moderator who can then accept it into the repository. Once a document has been accepted it will be visible to users by browsing or search the repository.
- *Removing existing documents*
  Documents can be permanently removed from the repository. Once a document has been removed all entries in the repository database and content files are removed from the server.

3.3.3.8.3 **Directory management**

A moderator has the ability to alter the hierarchy representing the repository by adding or removing directories. New directories are added by simply supplying the folder name, directory title and required security role for the new directory. Lastly, the location in the hierarchy of the new directory has to be selected from the list of existing directories. If a new directory is created with, for instance, a security role of USER and its parent directory has a WORKER role, the directory's role will automatically be changed to WORKER because of the security role cascading rules.

Existing directories can be removed from the hierarchy as well as all files contained within the directory. A moderator also has the choice of leaving the directory's files on the server for future use. In this case on the database entry for the directory in the directory table is removed to restrict users from browsing the contents of the deleted directory.

3.3.3.9 ***Configuration***

All relevant configuration information for the repository module is stored in its associated XML configuration file. The configuration manager is responsible for loading the initial configuration values and then monitoring the configuration file to ensure that when changes are made they are automatically loaded to ensure validity of configuration information. See Appendix A for an example configuration file for the repository module. .

3.3.3.9.1 **Standard settings**

Standard settings are settings required by the framework to allow for successful integration with the core components and other modules. The standard settings for the repository module is as follows:

- *Events & Plugins*
  Events are triggered by various actions within the module, for instance the addition of a new document. For each of the events there is a collection of associated plugins that are executed to allow for customisation of the module.
- *Navigation*
  The navigation sections contain a collection of links used by the JSP of the module. Each link consists of a type, URI, title, required condition and optional JavaScript code.
- *Help*
  Context sensitive help information is defined here, which is used by the JSP and module to present help on various actions.

3.3.3.9.2 **Additional settings**

In addition to the required standard settings, the module also relies on additional settings to be supplied. Additional settings are module specific, and consists of the following:

- *Database*
  The database settings contain information on the creation of the SQL database and also any default values to be inserted into the database. Basically, this section defines a set of SQL statements that can be used by an administrator when installing a new instance of the framework.
- *Repositories*
  Each repository instance is defined in the repository configuration. A repository is

described by an identifier, title, description and default required security role. Also, each of the available actions is assigned a security role to restrict access to that action to authorised users.

- *Metadata*
  Metadata is used to describe the properties of each of the documents contained within the repository. Each repository instance can have its own set of metadata fields describing the documents. This section defines the set of metadata fields used to describe the documents for each repository instance.
- *Open Archives Initiatives*
  This section contains a string commonly used by the OAI class for the repository. An XML node consisting of a name and the associated string represents each entry.
- *Search information*
  The search settings comprise of a list of supported document formats for indexing and various rendering options for search results. Search results can be customized without altering the underlying code by separating the formatting options and the actual Java code.
- *Document downloading options*
  As with the search results, the rendering of a document's downloading options is also defined in the repository's XML configuration document. Various downloading options for a document's content exist, and can be customized for each of the repository instances. For example, the workspace repository allows a document to be checked-out for editing.
- *Validators*
  Validators are used to ensure the validity of information supplied by the user when, for instance, a new page is submitted. This section contains a collection of Validators each representing an HTML form field. The Validators are grouped together to allow for selective use, based on the current action. For example, there exists a group of Validators associated with submitting a new page.

3.3.4 Statement database

3.3.4.1 ***Overview***

The statement database allows users to make statements or submit quotes regarding certain topics. Each topic database contains a collection of statements that are grouped by the creator of the statement. Statements are described using a reduced Dublin Core metadata set consisting of a date, creator, rights and original source. Additional metadata fields that were added include the person who submitted the statement and any additional notes.

The statement database can be used as a general database where users can post ideas or comments regarding certain topics. These statements can then be discussed further using, for example, the online discussion forum and eventually a new research project might develop.

*3.3.4.2 **Classes***

The following collection of Java classes were created to form the statement database module.

3.3.4.2.1 **Module**

The module class is found throughout the framework, and is essentially a specialization of the root Module class that defines and exposes certain required functionalities. Most of the required functionalities have already been defined in the root Module class, thus requiring very little specialization of the class.

An example of specializing the module is the loading of security roles associated with each action. The availability of actions to users is based on security constraints, for example only administrators can remove statements from the database.

3.3.4.2.2 **Loader**

The loader is also a specialization of the root Loader class and is primarily concerned with loading initial configuration values, such as security roles, from the configuration files. A thread runs in the background to assure that any changes made to the module's associated configuration information is picked up and handled accordingly.

3.3.4.2.3 **Link**

Links are used to create references between either two statements or any other resource available within the framework. A link consists of the following properties:

- *Identifier*
  Each link is identified by a unique identifier consisting of the topic and a unique numerical value.
- *Type*

Links can either point to a resource within the framework or to a URI outside of the scope of the framework.

- *Resource type*
  An internal link can point to either of the following resources: a document, a message in the discussion forum, another statement or a page.
- *URI*
  The URI of the end point is created based on the type of resource and the scope of the link. If, for instance, the link points to a document in the digital library the link will contain the repository instance name and the document's unique identifier.
- *Title*
  The title of the link can either be the title of the web page to which the link points or the title of the resource, for instance the title of a document in the digital library.
- *Internal document identifier*
  If the link points to an internal resource, the internal document identifier will be the unique identifier of the resource to which the link points, for instance the unique identifier of a document in the digital library.

3.3.4.2.4 **Metadata**

Metadata is used to describe the properties and attributes of a statement. In essence, the metadata class consists of a collection of fields that can be used to describe a resource. The field information is loaded for the module's configuration file and stored in a hash table to allow for lookups.

In addition to containing a list of all the fields, the metadata class also assists in ensuring that required information is supplied when submitting a new document by utilizing the Validator component.

3.3.4.2.5 **Statement**

Statement information is stored in a database and also a serialized Statement object file. The database contains the metadata information describing a link and the serialized object contains a list of all the statement links. When the list of statements is compiled for the user, only the metadata from the database is loaded; however, when a user views a specific statement the metadata as well as the collection of links is loaded. This is done because the links are not required when browsing the collection of links, thus saving processing power.

3.3.4.2.6 **StatementDatabase**

A statement database is used to group statements covering the same topic. Each database is described by a title, description and collection of weighted theme values. Also, security constraints are placed on each available action, for instance only administrators can remove a statement from the database.

Statement databases are defined in the module’s associated XML configuration document, which is described in Section 3.3.4.7.

3.3.4.3 ***Integration***

One of the main goals of developing a modular framework was to allow various models to interact with one another and also with the core functionalities of the framework. For example, the repository module uses the interface functionalities exposed by the core Interface component.

The interaction between the statement database module and other modules and components contained within the framework are similar to that of the content module as described in Section 3.3.1.

3.3.4.4 ***Presentation & Functionalities***

3.3.4.4.1 **Browsing databases**

There can exist various databases each representing a certain topic and containing a collection of statements. A user can select a specific database form the database collection or by following a link from another resource. Each database in the list is described by a title, description and a collection of weighted theme values.

After selecting a database, the user is presented with the collection of statements contained within the database. Statements are grouped according to the creator and then sorted by date. Each entry in the list represents a statement and comprises of the actual statement and its date of submission. A user can then choose to view more detailed information on any of the presented statements.

3.3.4.4.2 **Viewing statements**

When viewing a statement the user is presented with the statement text as well as the option to view any of the associated metadata describing the statement. The metadata describing the statement consists of submission date, creator, original source, additional notes, copyright information and the person who submitted the statement.

In addition to the viewing the statement's text and metadata, a user can also view any related links to additional resources that were created by other users. Also, a user can create his own link to any resource within the framework or to a web site outside of the framework's scope. Links are created using a wizard and requires a user to complete a set of steps in order to gather the required link information. Section 3.3.1.2.4 describes the Link object that represents each link within the statement's collection.

3.3.4.5 ***Database***

A database is used to store metadata associated with each of the statements contained within each of databases. There exists only one database table for all of the statements and the malcolm_database field is responsible for indicating the database to which a statement belongs.

Field name	*Field type*	*Field description*
dc_identifier	varchar(36)	A unique identifier for each statement.
malcolm_statement	blob	The actual statement text.
dc_creator	varchar(100)	The creator of the statement.
dc_date	date	Date of submission or when statement was made.
dc_source	blob	Original source of the statement.
dc_rights	blob	Copyright information associated with statement.
malcolm_submitted By	varchar(16)	Person who submitted the statement.
malcolm_accepted	tinyint(1)	Boolean value indicating whether statement has been accepted.
malcolm_notes	blob	Any additional notes on the statement.
malcolm_database	varchar(50)	Name of the database to which the statement belongs.

Table 4.9. Definition of the Statement databases' table.

3.3.4.6 ***Statement database management***

A limited set of management options is available to users of the statement database. Certain management actions are limited to authorized users only. The three management actions are the following:

- *Create a new statement*
  Most users will be able to create a new statement in a database. Users are required to fill in the required metadata information in order to submit a new statement. Validators are used to ensure that required fields are filled in and that

the data is in the correct format.

- *Accept a pending statement*
  Once a new statement has been submitted a moderator must first accept it before it will be visible to general users. A moderator can list all the statements within a selected database and then accept or reject pending statements.
- *Remove an existing statement*
  A moderator can select statements to be removed from the statement databases. Once a statement has been deleted its metadata will be removed from the database as well as its collection of links.

3.3.4.7 **Configuration**

All relevant configuration information for the statement database module is stored in its associated XML configuration file. The configuration manager is responsible for loading the initial configuration values and then monitoring the configuration file to ensure that when changes are made they are automatically loaded to ensure validity of configuration information. See Appendix A for an example configuration file for the statement database module.

3.3.4.7.1 **Standard settings**

Standard settings are settings required by the framework to allow for successful integration with the core components and other modules. The standard settings for the repository module is as follows:

- *Events & Plugins*
  Events are triggered by various actions within the module, for instance the addition of a new document. For each of the events there is a collection of associated plugins that are executed to allow for customisation of the module.
- *Navigation*
  The navigation sections contain a collection of links used by the JSP of the module. Each link consists of a type, URI, title, required condition and optional JavaScript code.
- *Help*
  Context sensitive help information is defined here, which is used by the JSP and module to present help on various actions.

3.3.4.7.2 **Additional settings**

In addition to the required standard settings, the module also relies on additional settings to be supplied. Additional settings are module specific, and consists of the following:

- *Database*
  The database settings contain information on the creation of the SQL database and also any default values to be inserted into the database. Basically, this section defines a set of SQL statements that can be used by an administrator when installing a new instance of the framework.
- *Statement databases*
  This section described the collection of statement databases. Each database has a title, description and a set of themes describing it. Also, security constraints are defined for each of the actions, for instance removing an existing statement from the database.
- *Metadata*
  Metadata is used to describe the properties of each of the statements contained within the databases. This section defines the set of metadata fields used to describe the statements for each of the databases.
- *Browsing*
  The various methods of rendering a statement is defined in the browsing section. This allows administrators to alter the presentation of statements without the need to edit the underlying Java code. The three rendering options that can be altered are rendering statements for users browsing a database, administrators accepting statements and administrators removing existing statements.
- *Validators*
  Validators are used to ensure the validity of information supplied by the user when, for instance, a new page is submitted. This section contains a collection of Validators each representing an HTML form field. The Validators are grouped together to allow for selective use, based on the current action. For example, there exists a group of Validators associated with submitting a new statement.

3.4 ***Supporting core modules***

In addition to the various functional components that were described in the preceding section, the framework also consists of a collection of core components. These components provide the base for the development of the framework and are not necessarily always accessible or apparent to the users.

3.4.1 Graphical User Interface component

3.4.1.1 ***Overview***

A key aspect of the framework was to allow for the modification of the interface without altering the underlying Java code. This was achieved through the use of XML and XSL documents. Each component and module represents its output using XML and the interface component is responsible for transforming the XML to HTML using a set of XSL documents.

By separating the presentation from the processing of a module's output the resulting HTML can be customized to suite the user's personal preferences as per the requirements of the framework.

3.4.1.2 ***Classes***

The GUI component's classes are used extensively by each of the modules' JSP pages to present the user with content. Various classes were developed to assists in the rendering of HTML and the presentation of navigational links.

3.4.1.2.1 **Loader**

The loader is also a specialization of the root Loader class and is primarily concerned with loading initial configuration values, such as subjects, from the configuration files. A thread runs in the background to assure that any changes made to the module's associated configuration information is picked up and handled accordingly.

3.4.1.2.2 **FieldValidator**

Validators are used throughout the framework to ensure that user input is valid. The FieldValidator class assists in validating HTML input by using any of the following methods:

- *Required*
  A Validator can be used to check whether required fields contain a value. No checking is done, however, for the validity of the field's value.
- *Regular expression*
  Certain fields' values have to be in the correct format to ensure data validity. Regular expressions can be used to validate the input of a field to check whether it is in the correct format. For example, if the value of a date field has to be in the

format yyyy-mm-dd, then a Validator is used with the following regular expression: ^\d{4}\-\d{2}\-\d{2}$.

- *Compare*
  The compare Validator is used to compare the values of two or more fields. An example use is when a user has to enter his password twice when registering as a new user.

Validators are defined in each of the modules' and components' configuration documents and are grouped according to actions. For example, the repository module has a group of Validators that are used when a new document is submitted. A module is responsible for retrieving the group of Validators for the specific action after which each of the Validators is inserted into the HTML as JavaScript. The JavaScript is responsible for validating the values supplied by the user when the HTML form is submitted. An error message is displayed for each of the fields containing an incorrect value.

3.4.1.2.3 **Link**

All navigational links used within the framework are defined in a module's XML configuration document and is represented by a Link object. Links are divided into primary and secondary links depending on whether they are intended for primary or secondary navigational purposes. In order to allow for flexibility parameters can be passed to links, which are then used to substitute variables in the URI's.

Each link is represented in the configuration document by an XML node containing the following:

- *Type*
  The type of link can either be primary or secondary.
- *URI*
  A link can either point to another resource within the framework, a URL of a web site or a JavaScript function. JavaScript functions can be defined in the Link's script block.
- *Title*
  The title or label of the link that is rendered in the HTML or optionally it can be an image.
- *Conditions*
  Links are displayed conditionally based on certain criteria that are defined in the conditions block. Conditions consist of a required user role and optional actions.

- *Script*
  Any additional JavaScript functions performed by the Link must be defined in the script block. The Link's JavaScript functions will be inserted in the HTML document's script element.

A Link object "knows" whether it should be displayed based on certain criteria, such as the user's security role and the current action of the user. Before any link is rendered it first performs the following steps based on the current user's security role and action:

1. *Check security role*
   Each link requires a user to have a specific security role before it will be accessible to the user.
2. *Check conditional constraints*
   A link will only be displayed if the user is performing the correct action. For example, a link might only be required for when a user is managing the group of users. Thus a constraint can be used to check whether the user's current action is 'manage_users'. An Expression object, which is described in Section 3.4.1.2.4, is used to represent each constraint.

3.4.1.2.4 **Expression**

Expressions are used to compare the user's current action with the constraints of a specific link to determine whether a link should be rendered. An expression can compare parameters with pre-defined values and can either be required to match a certain value or not to match a value. There is no limit on the number of expressions that can be attached to a link and a link will only be rendered if all of the conditions are met.

For example, the link to add a new user is only required when a user is managing the group of users within the framework; an expression can then be used to check whether the user's current action is 'manage_users' and then only render the link if the match was successful.

3.4.1.2.5 **Interface**

The Interface class is responsible for managing the XML output of a JSP page and transforming it to HTML using a selected XSL document. Each JSP page is responsible for loading the appropriate Interface from the framework and passing XML and any additional parameters to the object. Link management is also done by the Interface, which is responsible for loading the appropriate links based on the current user's security role and current action.

Interfaces are defined in the component's associated XML configuration document and are loaded by the Interface component's Loader. An Interface consists of a collection of components, metadata, and XSL transformer. An Interface exposes certain components that are defined in its associated XSL document that can be accessed by the JSP page. Each component is in essence an XML fragment that is rendered by the XSL transformer resulting in an HTML document.

An additional function of the Interface class is link management. Links are defined in a module's associated XML configuration document, which are then loaded by the module and stored internally. The Interface class assists in determining which links to load by exposing a function that returns a set of links conforming to certain criteria. For instance, the JSP page passes the current user's security role and action to the Interface, which will then query the current module and retrieve the appropriate links. The collection of links is then passed to the Interface, which in turn renders the links to be presented as HTML.

3.4.1.2.6 **InterfaceComponent**

An Interface consists of a collection of components represented by InterfaceComponent objects. Each object contains an XML fragment that can be altered and rendered as HTML by the XSL transformer.

*3.4.1.3 **XML & XSL***

A combination of XML and XSL is used in order to allow for multiple presentations of the same content without the need to alter the underlying Java code. Each module's output is represented using XML, which is used by the JSP page in conjunction with an Interface object to produce HTML. An XML representation of a module's output consists of the following:

- *Metadata*
  Metadata can be used to describe the HTML document for a number of reasons. One such reason is for the search engine to better index the content of a page. The metadata section consists of a collection of metadata fields each with a name and value. An example of metadata for a page could be the title and author of the document.
- *Components*
  An HTML page comprises of a set of components, each representing an area of

the document. Typical components include primary and secondary navigation, content sections and a footer. A component is an XML fragment that has a name and content, which is rendered using the Interface's XSL transformer.

The Interface's XSL document is responsible for transforming the XML components and metadata to HTML. This requires the XSL document to conform to the framework's standard functionalities to ensure that the information is correctly rendered. There exist three main templates that are responsible for transforming metadata, content and navigational links.

Examples of both the XML and XSL documents can be found in Appendix A.

3.4.1.4 ***JSP template***

JSP pages have to conform to a certain template to ensure security and link integrity. The JSP template consists of the following required components:

1. *Load user object*
   The username of the current user is stored in the JSP session, which is used to load the User object and profile for the current user. If the username is not defined the user will have to be redirected to the login page.
2. *Obtain parameters*
   All of the parameters passed to the JSP page must be obtained and stored in both an array and hash table for use by the security module and interface module. Example parameters include the current action, step and identifiers.
3. *Check security*
   Security works by passing the current user's security role and the current action of the user to the module being accessed. The module will then query the underlying security component to determine whether the user has the required credentials to perform the requested action. If the authorization fails the user will be redirected to a JSP page informing him/her of the failure.
4. *Load interface*
   After a user has been granted access to the JSP page the next step is to load the appropriate interface based on the module's settings and the user's personal profile. A copy of the Interface object is retrieved, which is used to populate the components that make out the interface.
5. *Load root links for top navigation*
   The framework contains a set of root links that are loaded by default, which must

be displayed in the appropriate component of the interface. These links are also constrained by security restrictions, which will be evaluated by the interface object.

6. *Populate interface components*
   Interface components have to be populated with the appropriate XML fragments to construct the content of the JSP page. Depending on the requirements of the module, certain components will be populated whereas others will be left empty. The interface will use the populated components to construct the final HTML page.
7. *Load primary navigation for module*
   The primary navigation for a module is loaded from the module’s XML configuration document and is identified by links of type PRIMARY. By default there exist three components for the primary links that can be accessed by the interface, viz. to the top,left and right of the page. It is therefore possible to place the navigation of the site on any of these locations, depending on the client's needs..
8. *Load secondary navigation for module*
   Secondary links are loaded and handled in the same manner as primary links and are identified by links of type SECONDARY.
9. *Render interface*
   After populating the navigation and content of the JSP page the interface object renders the collection of components as HTML and passes it the to the client’s browser.

3.4.1.5 ***Configuration***

The configuration for the GUI component is mainly concerned with defining the various interfaces available to the framework. Also, additional links and Validators that need to be loaded are listed in the configuration document.

3.4.1.5.1 **Standard settings**

The standard settings of the GUI component consist only of the plugins defined for various events that are triggered by the interface functions.

- *Events & Plugins*
  Events are triggered by various actions within the module, for instance the addition of a new document. For each of the events there is a collection of

associated plugins that are executed to allow for customisation of the module. An example list of such plugins include the notification email sent to a moderator in the event of changes that were made to a repository, or in the event that a user replies to a discussion forum post the user who posted the original entry is notified.

3.4.1.5.2 **Additional settings**

In addition to the required standard settings, this component also relies on additional settings to be supplied. Additional settings are module specific, and consists of the following:

- *Link loaders*
  Any additional links that need to be loaded for the framework are defined in this section. For example, the User component isn't responsible for loading its links, which requires the GUI component to do it instead.
- *Validator loaders*
  As with the loading of additional links, the GUI component must also load any additional Validators required by components, such as the list of Validators for the Users component.
- *Interfaces*
  All available interfaces must be defined in the GUI configuration document. Each interface consists of a name, path to an XSL document and a Boolean indicating whether the interface is available to users.

3.4.2 User component

3.4.2.1 ***Overview***

A key functionality of the framework is the ability to build a user profile that describes a user's individual interests and contains personal information. Each user is represented by a User component containing his personal details, history and personal preferences. The User component assists in the handling of all user related functionalities and is available to the rest of the framework's modules and components. For example, the user's model can be queried to obtain his list of personal preferences, which can be used to provide him/her with recommendations.

A relational database is used to store a user's personal details, such as his name, email address and list of memberships. This information is thus easily obtainable using standard SQL queries by the Java code. In addition to a database, a user's personal model is stored as a serialized User object and is stored on the server. The User object contains information relating to the user's history and personal preferences, which can be obtained through the User model's exposed functions.

3.4.2.2 ***Classes***

Three core Java classes, that expose certain management and querying functionalities, are used to define a user model. These classes are capable of interacting with other modules and the list of core components of the framework.

3.4.2.2.1 **Loader**

The loader is also a specialization of the root Loader class and is primarily concerned with loading initial configuration values, such as security roles, from the configuration files. A thread runs in the background to assure that any changes made to the module's associated configuration information is picked up and handled accordingly.

3.4.2.2.2 **ProfileEntry**

A user's history comprises of a collection of ProfileEntry objects, each containing the identifier of the resource and dates on which the resource was visited. The collection of profile entries can be queried to obtain the dates on which a resource has been visited in order to recommend additional resources to the users. This information can be presented to a user when he/she points to a link, which can assist a user in his decision making process. There exist four methods of querying the dates of a profile entry, viz.:

- *GetDates*
  Returns the list of unedited dates on which a user has visited a specific resource. The list may contain duplicate date entries. This is useful when calculating the number of times that a user has viewed a specific resource which in turn can be used to assess the popularity of that resource.
- *GetNumberOfVisitsForDate*
  Returns the number of times a user has visited a resource on a specific date. This can be used to determine trends throughout the population of users who access the information resources. By analysing trends, a moderator or site administrator can provide more accurate time-based recommendation to users.
- *GetDistinctDates*
  Returns a list of dates on which a user has visited a resource, with the list

containing no duplicate dates. This information can be useful to users of the services, by providing them with a list of dates on which they have accessed certain resources. A user could therefore make an informed decision on whether to access the resource or not.

- *GetDatesAndCount*
  Returns the list of distinct dates on which a user has visited a resource with the number of visits for each date. A user can use this information when deciding on whether to access a resource, because he/she now knows how often he/she has accessed each of the available resources.

3.4.2.2.3 **Profile**

A user profile is responsible for maintaining the user's personal preferences and history of resources that were visited. After each visit to a resource, the user's history is updated to reflect the new addition and modifications are made to the user's personal preferences. The Profile class exposes the functions required to maintain and query the user's profile. These functions include:

- *Loading properties*
  A user's details are stored in both a relational database and a serialized object. Depending on the information that is required, the details can either be loaded from the database or the serialized object. If only the user's personal details are required then the database will be used, otherwise for user's preferences the serialized object will be loaded.
- *Query history*
  The complete history of a user can be queried or a subset thereof. For example, the dates on which a user has visited a specific resource can be obtained or the complete list of dates for all of the resources that a user has visited. This information can then be used to recommend additional resources or be presented to an administrator to monitor site usage.
- *Update preferences*
  A user's personal preferences consist of a collection of weighted theme values that can be queried and updated using the user's profile. After each visit to a resource that is described by a ThemeStructure, the user's personal preferences are updated to reflect the user's changing interests. A combination of the time spent visiting the resource and the resource's weighted theme values are used to

increment a user's personal preferences.

- *Log entry*
  The user's history is maintained by logging each visit to a resource. If there exists no prior entry for the resource in the user's history a new entry will be made using the resource's unique identifier, type and the current date. Any subsequent visits to the resources will result in the addition of the current date to the list of dates on which the resource has been visited.
- *Update clusters*
  Each user belongs to at least one user cluster that is maintained by the User component's associated ClusterManager. After each update of a user's personal preferences the ClusterManager will re-evaluate the user's preferences and relocate the user to another cluster if the user's interests no longer fall within the current clusters.

3.4.2.2.4 **User**

The User class encapsulates all the personal information, preferences and functionalities of a user within the framework. Each user of the system is represented by a User object that is serialized to disk to allow for persistence. The main properties of the User class consist of the following:

- *Personal information*
  Contains the user's name, email address, password and Boolean value indicating whether his profile is private or public. This information is stored in the relational database as well as the serialized object representing the user.
- *Profile*
  An instance of the UserProfile class that contains the user's history and personal preferences. This object is serialized together with the personal information of the user.
- *Memberships*
  A hash table of all the research projects to which the user belongs together with the security role for each of the projects. For example, a user may have a WORKER security role for the AH research project and ADMIN for the WWW research project.

The User class also provides methods to allow for user registration, login and maintenance of personal information.

3.4.2.3 ***Functionalities***

There exists a broad range of functionalities available to the user via his personal homepage. Once a user has successfully logged in, he/she is directed to his personal homepage where he/she can perform any of the following functionalities.

3.4.2.3.1 **My homepage**

On a user's personal homepage a user is presented with the following blocks of information, viz.:

- *Pages*

  A list of recommended pages is displayed based on the user's personal preferences. The pages are grouped according to their respective subjects and are linked to the full content of the page. This provides a user with a recommended reading list every time he/she visits his homepage.

- *Forum topics*

  As with the recommended list of pages, the user is presented with a list of recommended forum topics. Each recommended topic consists of the title and description of the topic with a link to the discussion forum.

- *My documents*

  This section provides the user with a list of all the documents that he/she has submitted to any of the repositories in the framework. Documents are grouped according to their respective repositories and consist of the title, author and date of submission.

3.4.2.3.2 **View profile**

A user's profile is presented to him/her and allows for modifications to his personal information and preferences. The profile information is divided into the following three sections:

- *Personal details*

  A user can edit his name and email address by changing the values in the pop-up window.

- *Personal preferences*

  The values of each of the topics that describe the user's personal preferences can be altered manually by the user. The user is presented with a list of all the topics with the current weighted value together with all the weight options in a drop down menu. The values can then be modified and updated to reflect the user's level of interest for each of the topics.

- *History*

  A user’s history is grouped according to the modules contained within the framework. Each module’s history is represented by the collection of resources that were accessed together with the dates for each of the resources. A user can remove any item from the history if the framework administrator allows the modification of a user’s history.

3.4.2.3.3 **View recommendations**

This section contains recommendations based on a user’s personal preferences, history and similar users. Each of the three groups of recommendations is presented separately and is described further in Section 3.3.1.7.

3.4.2.3.4 **Create a new project**

The framework allows for various instances, each representing an individual research project. Project managers can request the addition of their projects to the framework by completing a project requisition form. The form consists of the project’s acronym and a short description. After submitting the form, the framework administrator can review the request and then create a new instance with the project manager as the administrator.

3.4.2.3.5 **Manage users**

Administrators of both the framework and individual projects can manage the collection of users and their rights. See Section 3.4.2.4 for the available management functionalities.

3.4.2.4 ***User management***

Administrators of both the framework and individual projects can manage the collection of users and their rights. Depending on the administrator’s privileges, an administrator can perform any of the following management functionalities:

- *Remove an existing user*

  A user can be removed entirely from the collection of users of the framework. Once a user has been removed, his personal information stored in the database as well as his serialized User object will be deleted.

- *Edit a user’s role*

  Each user has a specific role within the collection of research projects, for instance a user might be a WORKER in one project and a normal USER in another. Administrators have the ability to alter the security roles for a user for any of the instances of the framework.

- *Edit a user's memberships*
  There can exist various research projects, each represented by an instance of the framework. Users must thus be allocated membership to the appropriate research projects. A user is required to request the collection of memberships to which he/she wants to have access to. It is then up to the individual administrators of each of the projects or the framework administrator to allows or deny users access to selected research projects.
- *Build user clusters*
  User clusters are used to group similar users based on their personal preferences. An administrator can periodically re-build the clusters to ensure that there are no stray users or clusters within the ClusterManager.

3.4.2.5 ***Database***

A relational database is used to store personal information describing each of the users of the site. In addition to the database, a user's personal preferences are stored in a serialized User object.

Field name	*Field type*	*Field description*
username	varchar(50)	The unique username of the user.
password	varchar(16)	The encrypted password of the user.
email	varchar(50)	The email address of the user.
name	varchar(50)	The name of the user.
public	tinyint(1)	A Boolean value indicating whether the user has a private or public profile.
membershi ps	varchar(50)	A list of research projects and security roles for each, e.g. www.1,ah.3.
clusters	varchar(255)	A list of cluster identifiers to which the user belongs.

Table 6.10. Field definition for the Users table.

3.4.2.6 ***Configuration***

The configuration for the User component is mainly concerned with defining the set of links, plugins and Validators used by the component.

3.4.2.6.1 **Standard settings**

Standard settings are settings required by the framework to allow for successful integration with the core components and other modules. The standard settings for the

User component is as follows:

- *Events & Plugins*
  Events are triggered by various actions within the module, for instance the addition of a new document. For each of the events there is a collection of associated plugins that are executed to allow for customisation of the component.
- *Navigation*
  The navigation sections contain a collection of links used by the JSP of the module. Each link consists of a type, URI, title, required condition and optional JavaScript code.
- *Help*
  Context sensitive help information is defined here, which is used by the JSP and component to present help on various actions.

3.4.2.6.2 **Additional settings**

In addition to the required standard settings, the component also relies on additional settings to be supplied. Additional settings are module specific, and consists of the following:

- *Database*
  The database settings contain information on the creation of the SQL database and also any default values to be inserted into the database. Basically, this section defines a set of SQL statements that can be used by an administrator when installing a new instance of the framework.
- *Validators*
  Validators are used to ensure the validity of information supplied by the user when, for instance, a new page is submitted. This section contains a collection of Validators each representing an HTML form field. The Validators are grouped together to allow for selective use, based on the current action. For example, there exists a group of Validators associated with registering as a new user.

3.4.3 Plugins

3.4.3.1 ***Overview***

Plugins are used by the framework to allow for the customisation of certain events that are triggered by modules and components. Each module exposes a set of events that can accommodate the use of custom plugins. For example, when a new user registers the User component's register event is fired, which in turn will execute the set of plugins. Plugins can be used, amongst other things, to alert a repository moderator of any additions to the digital library.

A plug-in class is created for each plug-in and specializes the Plugin class that is defined as part of the core framework. The chain of plugins for a module is defined within its associated XML configuration document. A plug-in node consists of the following properties:

- *Name*
  A name is given to each of the plugins to allow for identification.
- *Classname*
  The fully qualified Java class name of the plug-in's Java class information. This information is used to instantiate the correct instance of the plug-in.
- *Type*
  A plug-in can either be executed at the beginning or the end of an event.
- *Parameters*
  Parameters can be passed to plugins to allow for customisation. Each parameter consists of a name and an associated value. The values of parameters can contain place holders for variable substitution by the plug-in.

3.4.3.2 ***Plugin chain***

A chain containing a set of plugins is created for each event exposed by a module. Plugins are executed in the order in which they appear in the chain and can either be executed at the beginning or end of an event. The plug-in chains are defined in a module's associated XML configuration document and are loaded and then stored in a hash table.

The following steps are taken for an event that requires the execution of plugins:

1. *Module loads appropriate chain*
   The current module loads the correct chain of plugins from the global chain manager.
2. *Event function passes parameters to chain*
   The current event then passes the current object and event type as parameters to the chain, which will be used during the execution of the plugins. Only plugins that match the 'type' parameter, PRE or POST, are executed. The object that is to be modified or containing information used by the plug-in is also passed to the

chain. Example objects are a document contained within the repository or a User object.

3. *Plugin chain loads appropriate plugins*
   Only the appropriate plugins are executed by the chain. As mentioned, plugins can either be executed at the beginning or the end of a function as specified by their type property.
4. *Execution of plugins*
   Lastly, the set of plugins is executed by passing the object to the plug-in for servicing. A plug-in can either edit the object that is passed to it, or use information contained within the object to perform additional tasks. For example, a document contained within the repository can be passed to a plug-in, which will use the title and author to alert the moderator of a new submission to a digital library.

3.4.3.3 ***Examples***

The following is a list of default plugins that have been developed and that can be customised to suit the requirements of the project.

3.4.3.3.1 **AlterMessage**

Any email message that is sent using the Utilities class will automatically be altered by this plug-in. Examples of how an email message can be altered includes adding a default header or footer to the message, scanning the email for viruses, redirecting the message to a central repository and so forth.

3.4.3.3.2 **NotifyAdministrator**

In the case of certain events, such as the addition of a new document, an administrator is notified via email of that event. Plugins can be customised to include information regarding the event, for instance the name of the new document and the person who has submitted the document. This plug-in thus allows the administrator to quickly act in the case of events that require his attention.

3.4.3.3.3 **AlterContent**

One of the framework's core functionalities is to allow users to submit content to the various repositories and discussion forums. For this reason it might be necessary to alter the content of information that is submitted to the workspace to ensure that inappropriate words are removed. This plug-in can be set-up to make use of a word list that contains words deemed inappropriate for the workspace.

3.4.3.3.4 **NotifyOriginalSender**

This plug-in is useful for the discussion forum when users have posted various messages and can not afford to spend time monitoring the threads for replies. Once a user replies to a message in the discussion forum the user who has posted the original message is notified via email about the reply and is provided a link to the thread to allow him/her to view the reply.

3.4.4 Security

3.4.4.1 ***Overview***

Security plays a key role in the development of the framework to ensure authorization and authentication of users and their actions throughout the site. A role-based security model, that contains five levels, is used as the base for the security of the framework. All users of the framework are required to register and login before being allowed access to the site. However, there exists a default account which will be assigned by default to all visitors of the site that will allow limited access to information and functionalities contained within the framework.

3.4.4.2 ***Security model***

3.4.4.2.1 **Role-based security model**

A role-based security model is used to restrict the actions of users based on their level within the security hierarchy. There exists a total of five roles within the framework, viz.:

3.4.4.2.1.1 **Default**

All visitors to the site are assigned the default role which will allow them to access static content contained within the site. This is the most basic account and is an anonymous account, hence users are not required to login to use the account. As a result of the visitor's anonymity no information regarding his navigation and preferences is stored, thus limiting the ability to provide any adaptive functionalities.

3.4.4.2.1.2 **User**

Almost all functionalities and modules within the framework requires a user to be registered. This security role will allow users to browse all static content and also allow access to most of the modules. However, users will in most cases not be able to submit any information to the site. These users will, for instance, consist of students that need to access information but has no need to submit new information to the site.

Any user that registers will be assigned this role, which only an administrator is able to change, and is required to login to be able to access this site. Registered users thus have the added ability to change preferences and all navigation information is logged and recommendations are made based on the user's preferences and navigational behaviour.

3.4.4.2.1.3 **Worker**

A worker is a user that has a more active role in the framework. Typically, this user is able to submit documents to the repositories, post messages in the forum and submit content to the site. These users will, for instance, consist of researchers and project leaders that need to publish information to the various repositories and discussion forums.

As with users, workers are also required to register and login before being able to use the site. Only an administrator can assign a user the worker role.

3.4.4.2.1.4 **Admin**

Administrators are allowed all the functionalities and actions of workers with the additional ability to accept information that has been placed in the pending queue. Also, administrators can edit the roles and memberships of all users and workers of the site.

As with users, administrators are also required to register and login before being able to use the site.

3.4.4.2.1.5 **Super**

A super user, in essence, is an administrator with the ability to grant a worker administrative privileges.

3.4.4.2.2 **Authorization**

Authorization of users is achieved by requiring all users of the site, except visitors, to first register and then login to access the site. Upon a user's first visit, he/she is required to register by providing certain required information, such as a username and password. The user will then be assigned the User role which, as mentioned in Section 3.4.4.2.1 will allow him/her certain privileges. On each returning visit, the user will have to supply his username and password in order to gain access to the site.

3.4.4.2.3 **Authentication**

Authentication is the process of ensuring the user who is attempting to perform an operation is who he/she claims to be. As security plays a key role in the framework, each operation that is exposed to the users of the site requires a certain security level. Thus, for each action that the user performs his current security level, which is stored in a session database, is compared to that of the current action. If the user has a sufficient security role he/she will be granted privileges to perform the requested action. In the case that the user is trying to perform an action for which he/she does not have a sufficient security role he/she will be redirected to a error page informing him/her of the access violation. By checking the user's credentials for each action the chance of URI spoofing is greatly reduced.

An additional means of ensuring authentication is by only displaying the links to actions for which the user has a sufficient security level. Section 3.4.4.3 discusses this in further detail.

3.4.4.3 ***Navigation***

XML configuration files are used throughout the framework to allow easy customization of components and their primary and secondary navigation. Each navigational component of the site contains data relating to the title and the destination of the link. In addition, each link also contains information that is used to determine whether the link should be rendered or not.

When a collection of links is to be displayed the Interface module will first perform the following two checks to determine whether a link should be rendered;

- *Applicability (is the link valid for the current action)*
  Firstly, each link's applicability is determined based on a set of tests that must all be true in order for the link to be valid for the current page that the user is viewing. For instance, the Manage Repository link will only be displayed as a secondary link if the user is currently viewing a repository. By limiting the links in this manner, a user is not able to perform actions that might cause the framework to behave in a manner that might compromise security.

- *Sufficient rights*
  Secondly, each link contains a minimum user role that the user must have in order to access the link. For instance, if a link requires the Worker role only Workers, Administrators and Super users will be able to see the link. This will thus also limit the links that a user will be able to see, thus limiting the ability of a user

to perform URL spoofing to perform actions because he/she will not be aware of the function in the first place.

3.5 **Research methodology**

3.5.1 Research approach

A development research approach was followed during the development and implementation of the framework. This section will provide an explanation of what development research entails as well as the most common methods thereof. Lastly, this section will provide an explanation as to why this research project could be considered a development research project.

3.5.1.1 ***What is development research and why do we use it?***

Development research, unlike traditional research, entails an iterative and interactive development approach where both the development team and the various stakeholders work closely together. This results in the participants being able to provide the development team with timely feedback on various aspects of the project. There exists various motives [van den Akker 1999] for the use of development research in projects, viz.

- Timely feedback
  In the case of a project being developed for a specific research group, development research assists the developers in obtaining timely feedback from the parties involved. This assists the team in identifying and correcting issues which might arise before such issues cause to a great impact on the rest of the system. For example, the proposed method of submitting information to the various repositories is too complex, according to the participants, then the development team could change the process and incorporate those lessons into the other components of the framework without the need to revise previous work.
- Evolutionary design
  It is nearly impossible to envision exactly how a system should function and which functionalities should be provided. Development research could therefore assist researchers and developers in constantly revising the specifications and requirements to incorporate the lessons that were learnt as well as new features that were identified. For example, during the development process, a new feature such as document versioning could be identified as a necessity for a collaborative workspace.

- Relevancy

  Too often research is conducted in seclusion and researchers lose their relevancy in the research community. By occasionally obtaining feedback from other researchers, a research project's team could re-evaluate their approach in solving a specific problem.

3.5.1.2 ***What are the most common attributes of development research?***

Development research does not necessarily differ that much from traditional research. There are, however, some specific features that are unique to development research [van den Akker 1999]. The two main groups are formative evaluation and problems associated with development research.

- *Formative evaluation*

  Development research projects would typically have less respondents in the evaluation process. This has the result that the evaluation process of the project provides results of a higher quality and relevance than that of a project following a more traditional approach. One reason for the higher quality feedback is due to the fact that the researcher can interview each participant individually thus obtaining more in depth information regarding the various aspects of the project.

- *Problems associated with development research*

  Role division is sometimes a problematic issue in development research projects. This is because of the conflicting views of the researchers and developers with regard to which features should be incorporated in to a project. For example, developers would usually pursue their ideals for creative innovation whereas researchers would opt for a more empirical motivation. Development researcher is sometimes also plagued by a lack of formal methodological processes. It might therefore be required, in certain events, to adopt the research approach in order to accommodate the changing of, for instance, the organisational structure of the project.

3.5.1.3 ***Why can this project be seen as development research?***

During the development of this project the user requirements were constantly refined. As a result, the project had to constantly evolve to meet the new specifications that were set out. During each iteration of the development process, the lessons that were learnt could be incorporated in the following phases. This resulted in a framework that ultimately met all the user requirements of the various parties involved, whilst incorporating all the lessons that were learnt during the various design phases in order to ensure a robust, effective and efficient framework.

3.5.2 Research method

The research methods that were used during the development of this project consisted of three main areas. Each of these areas attributed to the project by providing a motivation and method for the research, data collection, and interpretation of outcomes. This section will describe these areas in more detail and how each attributed to the final outcome of the project.

3.5.2.1 ***Sampling***

The motivation for the use of the DISSAnet project as a platform for the development of a framework was based on the client requirements set out by the project members. An online collaborative workspace was needed by the DISSAnet project which would enable the participants to share research findings and collaborate on various projects. This proved to be an ideal project to investigate the user requirements of such a system and also whether such a system could successfully be implemented in similar projects. The research participants were chosen based on their involvement in the DISSAnet project as well as which contributions they could make to the identification of the user requirements and usability testing of the final framework.

3.5.2.2 ***Data collection***

During the design stages and development of the framework, various methods and sources of data collection were utilised. It was imperative to collect data at each of the stages to ensure that the user requirements were met and also to ascertain whether be best approach was always followed. The three main stages of data collection were:

- *Interviews*
  The interview process consisted of an informal process where a discussion was held with various researchers and users of online services, such as workspaces, discussion forums and digital libraries. During the interview process, the user requirements for each of these systems were identified, discussed and documented. After the completion of the interview process, the various requirements for each of the components were used to construct a broad list of specifications that could used to develop a prototype system. Additional interviews were held during the various development phases to constantly obtain feedback which could then be incorporated during the following phases.

- *Usability testing*

  The usability testing of the framework involved working closely with the various participants project. There existed two main methods of assessing the usability of the framework, viz.

  - Interviews

    The current phase of the framework was presented to the various research participants after which an informal interview was conducted. During the interview process, the participant were allowed to ask questions and make comments regarding, amongst others, the steps involved in preforming certain tasks, the structuring of information and the interface. This information was then documented and incorporated during the following development phases.

  - Observation

    Another method of obtaining data relating to the usability of the framework was to observer participants whilst performing certain tasks. This process comprised of giving the participants a certain set of tasks which they had to perform. During this process, each participant's actions and responses were observed to determine where he/she had difficulty performing a task. This information was then also documented and provided an invaluable insight as to where the usability of the framework was lacking.

- *Development approach*

  The development approach that was followed required constant re-evaluation and revision of work that has already been done. Details of the various components and their respective features were constantly documented during the various development phases. Examples of such details include methods and their parameters, database tables and interaction with the user. This information was then used with the development of each new module to determine how the existing modules could be utilised and modified to accommodate the integration of the new module. Other examples of data that was collected was lessons that were learnt during the previous development phases and how this information could be used to avoid making certain mistakes again.

3.5.2.3 **Interpretation**

An effort was made to follow or use best practice during this research project. This section will provide explanations for the reasoning behind the use of the various methodologies during this research study.

- *Identification of user requirements*

  Before being able to develop a framework for a research project, one first had to determine the various user requirements. This was done by performing both a literature survey and informal interviews. The reasoning behind the use of a literature survey was because of the knowledge that could be obtained form the scholarly research currently currently being done on collaborative research. Although the literature survey provided invaluable insights into collaborative workspaces, additional knowledge had to be obtained on the exact user requirements of such an online collaborative workspace. The informal interviews provided just that.

- *Design specifications*

  It was envisioned that the framework being developed could be implemented in various research projects in the future. For that reason it would be required to be able to add new or modify the existing components of the framework. A modular approach was therefore the best approach to ensure that the modification of one component would not impact the rest of the framework. Moreover, certain implementations would not require all of the components, and would therefore require that only certain features be implemented. Once again, the modular approach would assist one in enabling certain components without disrupting the rest of the framework.

- *Development of the framework*

  As stated earlier, a developmental approach was followed for the development of the framework. The motivation for this approach was because of the ever changing user requirements for which it was required to constantly revisit past work to ensure conformance. By working closely with the stakeholders involved in the project, it was possible to address both issues and new user requirements as they were identified. The interactive and iterative approach that was followed was therefore best suited to the nature of the project.

- *Usability testing*

  The usability testing, as with the development, approach was done as an interactive and iterative process. During the development of the project, the stakeholders were involved in assessing the various attributes involved with usability. By following this interactive approach, usability issues were quickly identified and addressed which ensured that those issues would not arise during future phases of the project.

3.5.2.4 ***Limitations***

Due to the scope of this research study, certain limitations do exist. Examples of such limitations include the number of participants involved in the user requirements and usability testing. As a result, the initial framework that was developed does not necessarily meet the needs of the broader research community. Moreover, the usability testing, which involved expert users, might not accurately address all the usability needs of less proficient computer users. It must be stated that this framework was developed with various implementations in mind, and that the lessons that will be learnt from each implementation will be used in order to further enhance the features and usability of the original framework.

3.6 ***Summary***

This chapter highlighted the need for careful consideration of which technologies, architecture and design principles are to be used in the development of a robust and flexible collaborative workspace framework. Various modules were described that were needed to ensure that such a framework could be developed in a manner allowing for further enhancements and the addition of added functionalities.

A key design principle that was followed during the development of the framework was modularity. Modularity is essential in any framework that wishes to reap the benefit of code reuse and integration. During the development of the framework, it became evident that the various modules would benefit from using existing modules' features. It was therefore required to constantly revisit and re-engineer those classes so that they could be used seamlessly throughout the framework. One such example of where integration was used in the development of this framework is through the use of a standardised Metadata object which is used to describe the various information objects. Another such an example is the standard digital library and a collaborative workspace which is both provided by the Repository module which can adapt to suit the needs of the specific implementation.

If any framework is to be successful, it has to be tested thoroughly whilst still under development. However, the most effective method of testing reliability and flexibility and whether the framework satisfies the needs of a project's members is by implementing it in various projects. Chapter 4 will discuss the various projects in which this framework have been implemented and also the resulting changes that were made in order to accommodate the users' specific needs.

4 Chapter 4 - Implementations

4.1 *Introduction*

The development of the framework consisted of first identifying the broad set of features that needed to be implemented. During the various development phases of the framework, each feature was investigated in more detail and subsequently revised to meet the needs of the users. Usability played a significant role in the development phases of the framework, as the constant feedback from the research experts which were involved ensured that the resulting framework met their functional and usability requirements. All of the user requirements were met after the completion of the initial framework. It was only during interviews and discussions with other research projects that if became evident that certain features were still lacking. This provided a great opportunity, however, to test the customisability and modularity of the framework during the resulting implementations.

Following is a list of all the projects where the framework has successfully been implemented, including a short description of each project.

- *DISSAnet*
  DISSAnet is a research project which aims to create a platform for the advancement of Library and Information Science research in South Africa. For this project it was necessary to create a collection of repositories where conference proceedings for conferences organised by DISSAnet could be stored.
- *IKS*
  The Indigenous Knowledge Systems (IKS) project provides a database of experts, projects and resources aimed at documenting and preserving indigenous knowledge held by communities throughout Southern Africa. The focus of this project is to provide visitors to the site with a comprehensive digital library rather than being a collaborative workspace.
- *IFLA-KM*
  The IFLA-KM is the official site of the Standing Committee for Knowledge Management of the International Federation of Library Associations and Institutions, IFLA. It provides visitors with various repositories, including, amongst others, a collection of toolkits, a library of articles relating to knowledge management and a collection of activities organised by the committee.
- *The Wellness Firm*

The Wellness Firm is a company that provides wellness services and solutions to corporate clients. The services range from complete physical assessments to online Journalling with registered psychologists. With this implementation the framework was used to provide a corporate intranet for The Wellness Firm's employees. It consists of a set of repositories, an online discussion forum and an online calendar.

Each of these implementations will be discussed in the following steps:

- *Overview*
  The overview provides background information on the project to highlight the various domains where the framework can be successfully implemented. This will also provide a better understanding of the basis for the client requirements.
- *Client requirements*
  This section will provide an overview of the purpose of the implementation and then also any additional requirements which might arise from the interviews with the project members. After obtaining a better understanding of the various requirements, it was then possible to ascertain which customisations were required.
- *Customisations*
  After obtaining a better understanding of the client's requirements, the process of customising the framework could begin. This consisted of customising the repositories' metadata schemas, interface and also available features. In some cases, additional functionalities were required, which were also then developed and implemented during this phase.

*4.2 **Implementations***

In order to test the effectiveness and usability of any system it is important to subject it to user testing and evaluation. This chapter reviews each of the instances where the framework has implemented as well as the usability issues that were encountered with each of these implementations. Each implementation could easily be customised to meet the client's specific needs due to the framework's vast array of features and modular design. Primarily, the framework is used for research projects where team members need to publish documents and use the Internet to collaborate with one another. However, the adaptability of the framework has allowed it to also be implemented in commercial sites focussing on providing visitors a rich browsing experience by allowing its owners the ability to update its content and maintain a digital library of useful resources.

The framework's design specifications were drawn up after careful examination of existing digital libraries and online collaborative workspaces. This allowed the framework to provide a broad spectrum of modules and functionalities, from which each new implementation could be developed. The various project leaders for each project were questioned on their specific needs before actually implementation began. From these interviews, it became evident that not all of the features of the framework were required for each of the projects. Moreover, the framework did not necessarily meet all of the project leaders' requirement, hence the need for the development of customised functionalities for each implementation. In addition to developing new features for each implementation, the design for each were also customised to meet the corporate identity requirements for each project. For example, each project focussed on a different research area requiring a custom metadata schema for its repositories and discussion forum topics. In certain cases, it was necessary to add a custom module to satisfy the requirements of the project. An example is the calendar module that was implemented for the IFLA-KM project.

With each alteration and customisation of the various implementations of the framework, the original framework grew to accommodate more modules and features which greatly improved its effectiveness. The feedback and requests from users of the various implementations provided invaluable information during the revisions of the original framework which was constantly revised to accommodate the users' needs.

Following is a list of the research as well as commercial implementations of the framework. Customisation of each implementation included custom designs and features, metadata schemas used to store documents in the various repositories and the availability of features to the users. Each of the implementations will be reviewed and a description of the project, features and customisations will be provided.

4.2.1 DISSAnet

4.2.1.1 ***Overview***

DISSAnet is a research project which aims to create a platform for the advancement of Library and Information Science research in South Africa. For this project it was necessary to create a collection of repositories where conference proceedings for conferences organised by DISSAnet could be stored. Visitors to the site are allowed access to the publicly published conference proceedings whereas moderators have the responsibility of maintaining the collection of repositories.

Also worth noting is the fact that the framework was developed based on the needs of DISSAnet and grew from there. The DISSAnet website is therefore an example of a pure implementation of the framework.

4.2.1.2 ***Client requirements***

The DISSAnet project's requirements to provide an online repository for conference proceedings became the main objective for the development of this framework. An informal specification document was drawn up by the DISSAnet project leaders outlining the desired features for such an online digital library. Even though the development of an online digital library would satisfy the current needs of the project, it was decided that additional features should be developed that could greatly benefit the DISSAnet project leaders in the future.

At the time of writing, the current implementation of the DISSAnet web site does not incorporate all the features of the framework, but plans do exist to further extend the features provided to its visitors.

4.2.1.3 ***Features and functionalities***

Following is an outline of the components and features provided by the DISSAnet website.

- *Static information*

  Not all of the information contained within the site is generated dynamically which therefore requires a moderator to publish static content to the site using the administrative functions provided by the framework. Examples of such content includes conference information and a history of DISSAnet.

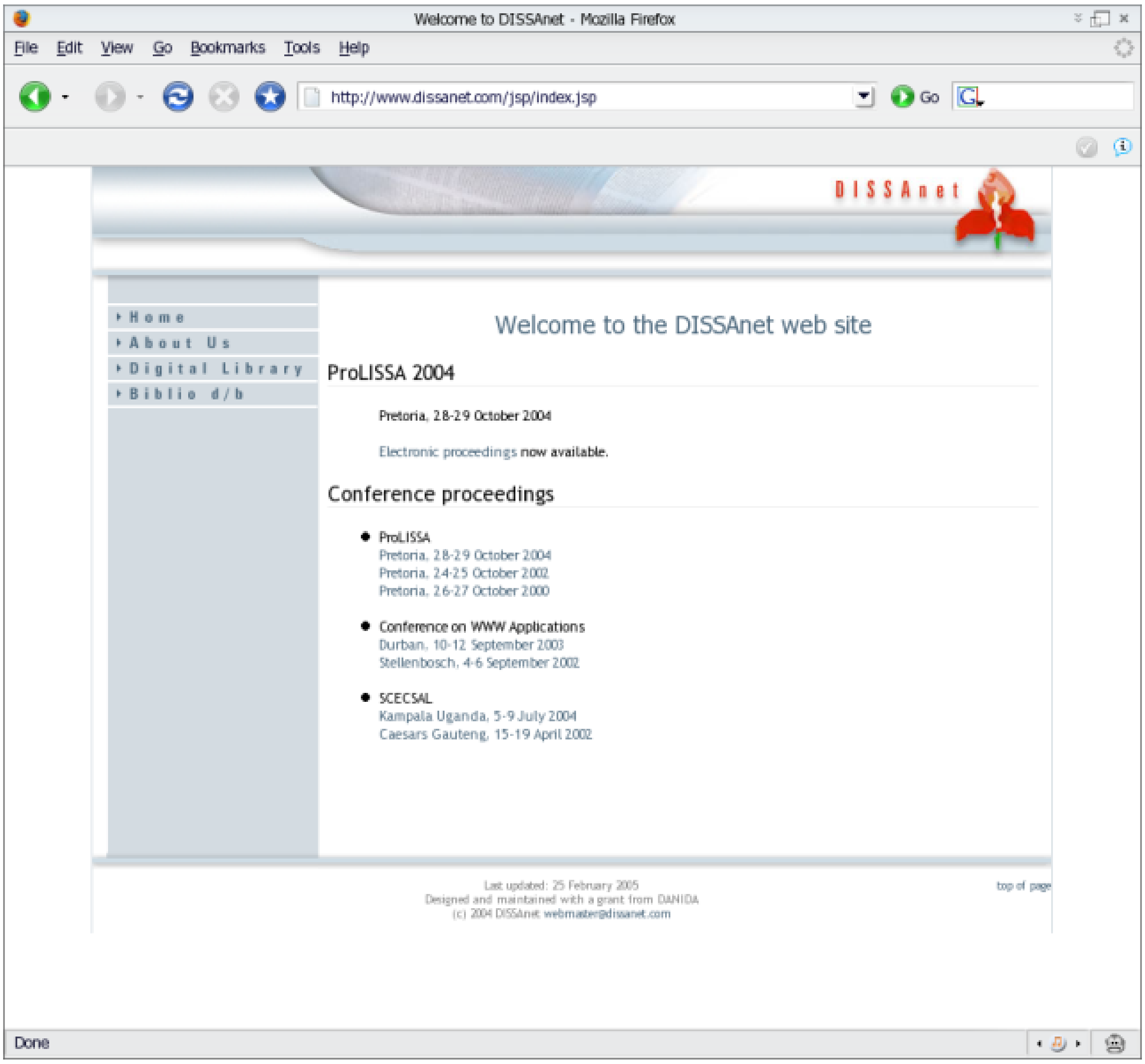
- *Browsing*

  Visitors to the site can browse for conference proceedings of the past conferences that were organised by DISSAnet as well as those of other conferences. The repository contains as its root folders the collection of conferences, each containing in turn a collection of dates on which the conferences were presented. For instance, a visitor could access the conference proceedings for the 2004 ProLISSA conference by selecting the ProLISSA conference folder and then following the link to the 2004 conference. Once a visitor has selected the desired conference, he/she will be presented with a collection of all the papers and presentations for that particular conference.

- *View conference proceeding*

After a visitor has selected the desired conference and date, he/she can then select a specific proceeding from the provided list. Each conference paper is accompanied by a full Dublin Core metadata schema describing, for instance, the title, author and date of the paper. In addition to the metadata, the visitor also has access to the full text of the paper in Adobe Acrobat format which he/she can either view in a frame below the metadata or in a new window.

- *Searching*
  Visitors can search for conference proceedings in cases where they don't have the detailed information relating to the conference or date. The search functionality provides visitors with both a simple and an advanced search option allowing them to provide as many or as few search parameters depending on the complexity of the search. Alternatively, visitors can perform a full text search by providing either a single word or sentence contained within the conference paper's full text document.
- *Repository management*
  Moderators of the site are tasked with managing the collection of conferences and their respective proceedings. This is done using the administrative functions provided by the framework. Moderators can create the top level folders for conferences and also sub-folders for each conference to allow for, for instance, the addition of a new date on which a specific conference took place. Moreover, moderators can manage the papers for each conference and can either submit a new paper or remove or edit an existing entry.
- *User management*
  The implementation of the framework only allows for two groups of users, viz. visitors and moderators. It is therefore the responsibility of the moderators alone to manage the repositories due to the fact that visitors to the site are limited to browsing information. There exists one user account which is used as a default account for all visitors to the site whilst moderators are required to login using their own user names and passwords.


Welcome to DISSAnet - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.dissanet.com/jsp/index.jsp
Go
DISSAnet
Home
About Us
Digital Library
Biblio d/b
Done

Welcome to the DISSAnet web site

ProLISSA 2004

Pretoria, 28-29 October 2004

Electronic proceedings now available.

Conference proceedings

- ProLISSA
  Pretoria, 28-29 October 2004
  Pretoria, 24-25 October 2002
  Pretoria, 26-27 October 2000
- Conference on WWW Applications
  Durban, 10-12 September 2003
  Stellenbosch, 4-6 September 2002
- SCECSAL
  Kampala Uganda, 5-9 July 2004
  Caesars Gauteng, 15-19 April 2002

Last updated: 25 February 2005
Designed and maintained with a grant from DANIDA
(c) 2004 DISSAnet webmaster@dissanet.com

top of page

Illustration 3: DISSAnet

4.2.1.4 ***Lessons learned***

The DISSAnet project was the first project for which the framework was implemented. During the implementation of the framework various lessons were learnt with respect to the user requirements, possible customisations and usability requirements of such a system. This section will reflect on the various lessons that were learnt and how they could affect possible future implementations of the framework in other projects.

- *User requirements*

  The user requirements of the system continually changed during the development of the framework. An important aspect to keep in mind during future implementations is that the user requirements could change over time which requires one to be flexible enough to adapt to these changes. From the implementation of the framework in the DISSAnet project, it was evident though that the modular design of the framework was the best design approach.

- *Customisations*
  Certain aspects of the framework had to be customised after the initial prototype had been developed and tested. These included the interface of the web site, the metadata schemas and also the modules that were utilised. The ease with which these customisations were possible is a testament to the flexibility of the framework.
- *Usability requirements and testing*
  The initial testing of the framework for the DISSAnet project highlighted certain difficulties when performing certain tasks. By evaluating the user response that was obtained during testing and interviews, one could quickly determine which areas of the framework required improvements with respect to usability. A great advantage of working closely with the project stakeholders was that user requirements, with respect to usability, were always identified and addressed promptly.

4.2.2 IKS

4.2.2.1 ***Overview***

The Indigenous Knowledge Systems (IKS) project provides a database of experts, projects and resources aimed at documenting and preserving indigenous knowledge held by communities throughout Southern Africa. Currently, the various repositories are maintained by the University of Cape Town's Centre for Information Literacy and a group of individuals. The IKS project's main areas of interest lie with documenting experts, projects and resources that were identified as being important in the understanding and preservation of indigenous knowledge. At the time of writing, there was a total of roughly 2900 records in the three repositories.

IKS's focus is more on providing an online digital library than being an online collaborative workspace. Members are able to upload records to the database and, pending a moderator's approval, the records will be made accessible to other members via either browsing and searching for resources. Similarly to the DISSAnet project, IKS also provides browsing, searching and management functionalities. However, unlike DISSAnet, IKS only allows registered members to access information through browsing and searching of the three repositories. In addition to the before mentioned functionalities, IKS also provides a discussion forum consisting of a four main threads, one for each repository, as well as a general discussion board. The forum allows members to submit new questions and statements or to reply on existing posts.

4.2.2.2 ***Client requirements***

The IKS project proved to be a great test case for the framework's ability to allow for various repositories where each repository uses a different custom metadata schema to define their respective information objects. Most digital libraries make use of standard metadata schemas, such as Dublin Core, to describe their information objects. However, the IKS project's aim is to describe information objects that aren't found in standard digital libraries. It quickly became evident that a new collection of metadata schemas had to be developed in order to accurately describe these information objects. For example, one of the repositories contains a collection of experts in their respective fields. Each expert is described using a custom metadata schema that was specifically designed for the IKS project to ensure not only an accurate description of the information object, but also its relations to other objects in the various repositories.

In addition to testing the frameworks adaptability with regards to metadata schemas, the IKS project also tested the framework's ability to store a large collection of information objects. The project required the storing, searching and browsing of thousands of information objects without impacting the usability and availability of the services.

4.2.2.3 ***Implementation and customisations***

As mentioned previously, the framework was designed in such a manner as so allow customisations as per the client's specific needs. One such an example is the modification of the browsing function, which for the projects repository added a map of South Africa with its provinces lighting up when the user points to one of the sub-folders. Moreover, when the user clicks on a specific province, he/she will be directed to that specific province's projects within the repository. These features were added to enhance the browsing experience for the user by providing multiple methods of accessing the same data in the repository hierarchy.

In addition to the customisation of functionalities, the IKS project also required custom metadata schemas to be used in order to accurately describe the various repositories' resources. Each repository had its own specific requirements, in addition to the Dublin Core metadata schema, with regard to metadata due to the specialised nature of the resources being described. As an example, consider the experts repository which is used to provide a database of experts within certain fields of indigenous knowledge. It was obvious from the start of the project that the standard metadata schema used to describe literature would not be sufficient and therefore close attention was paid when creating an additional schema to adequately describe these resources. Most of the new fields that were identified were used to describe biographical and contact information for the experts and contained simple free formed textual data. There were, however, certain fields that made use of lists to ensure data conformity when dealing with for instance research areas. These fields allowed for multiple selection of, for instance, research areas as well as cross-linking to projects contained within the projects repository. The following table lists the new metadata schema's set of custom fields.

Field name	***Description***
experts_title	The title of the expert, e.g. Mr.
experts_institutionName	The institution where the expert is employed.
experts_institutionType	The type of institution where the expert is employed.
experts_institutionPosition	Position held at institution.
experts_postalAddress	Postal address of expert.
experts_postalAddressCity	Postal address city.
experts_postalAddressCode	Postal code.
experts_physicalAddress	Physical address of the expert.
experts_physicalAddressCity	City where expert lives.
experts_physicalAddressCode	Physical address' code.
experts_tel	Telephone number of the expert.
experts_mobile	Mobile number of the expert.
experts_fax	Fax number of the expert.
experts_email	Email address of the expert.
experts_expertise	The expert's list of expertise.
experts_researchGroups	A list of research groups that the expert is involved in.
experts_projects	A list of projects that the expert is involved in.
experts_qualifications	A list of the expert's qualifications.
experts_uri	A URI to the expert's personal home page.

Table 11: Custom metadata fields for the Experts database.

Similarly to the experts database, the project database also required a customised metadata schema accurately describe the entries in the database. The projects metadata schema also consisted of a range of simple text fields such as contact information as well as lists to once again ensure data conformity. Below is a table containing the list of custom metadata fields that were identified.

Field name	***Description***
projects_dateCompleted	The date on which the project is to be completed.
projects_fieldPrimary	The primary fields with which the project is concerned.
projects_fieldSecondary	The secondary fields with which the project is concerned.
projects_fieldTertiary	The tertiary fields with which the project is concerned.
projects_keywords	A list of keywords describing the project.
projects_locationProvince	The province within South Africa where the project resides.
projects_locationRegion	The region within the province where the project resides.
projects_locationCoordinates	The GPS co-ordinates of the project's location.
projects_uri	A URI to the project's home page.
project_contactDetails	Contact details of the primary person responsible for the project.

Table 12: Custom metadata fields for the Projects database.

During the beginning stages of the project there weren't enough records for each repository to warrant the use of sub-folders. As a result, records were placed in the root of each the repositories. As the number of records grew, it became apparent that sub-folders were indeed needed for each repository. An additional feature was therefore added to allow moderators the ability to move existing records from the root of the repository, or any other sub-folder, to a different sub-folder contained within the same repository. This is an example of how the framework can be adapted so suit a client's needs and also how such features can be incorporated into the core framework enabling it to grow continuously.

4.2.2.4 ***Lessons learned***

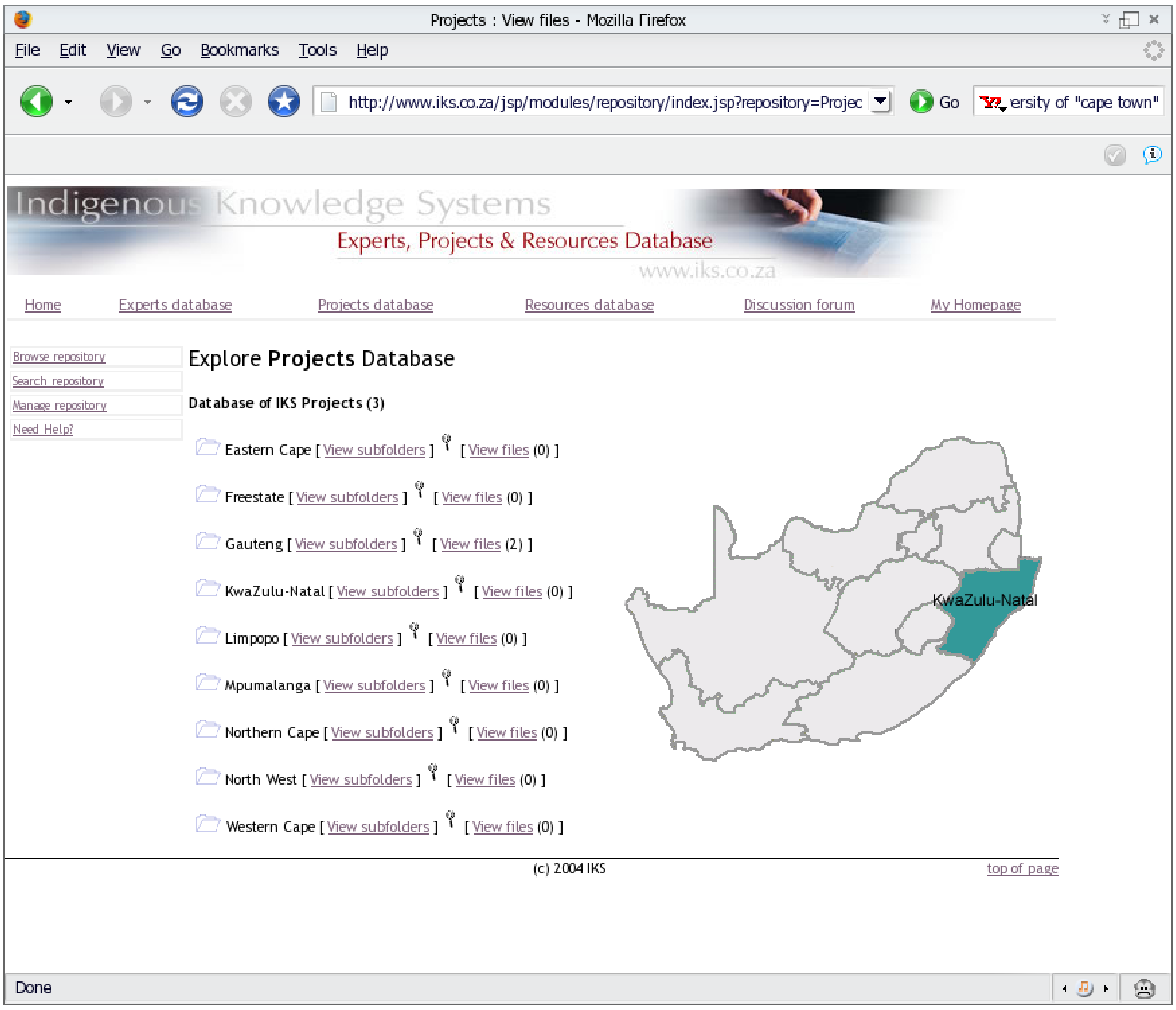
Various lessons were learnt during the initial implementation of the framework for the DISSAnet project. These lessons were then incorporated in the original framework to ensure that future implementations would benefit from those lessons. During the IKS project, however, certain new lessons and customisations arose. The following were learnt with respect to user requirements, customisations and usability requirements.

- *User requirements*

  The main focus of the IKS project's user requirements was to be able to have customised metadata schemas that could effectively describe the indigenous knowledge of the various repositories. During the later implementation phases certain changes had to be made to fields describing the various information objects. From this it was clear that ample time be spent during the initial phases of the implementation to ensure that the various repositories and their respective metadata schemas are defined correctly.

- *Customisations*
  The main customisations for the IKS project were for the development of customised metadata schemas. Fortunately the framework's support for custom metadata schemas succeeded in meeting the project's complex needs. Additional customisations were, however, required later on due to the increasing number of records that were stored in the various repositories. From this, it was evident that even though the initial user requirements are laid out during the initial phases of the implementation it could still be required to perform additional customisations during the various phases of the project.

- *Usability requirements*
  As with the DISSAnet project, the usability requirements of the project were addressed during the various phases by working closely with the various stakeholders. This close relationship, once again, proved to be very beneficial in the identification and addressing of usability requirements.


Projects : View files - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://www.iks.co.za/jsp/modules/repository/index.jsp?repository=Projec

Indigenous Knowledge Systems
Experts, Projects & Resources Database
www.iks.co.za

Home | Experts database | Projects database | Resources database | Discussion forum | My Homepage

Browse repository
Search repository
Manage repository
Need Help?

Explore Projects Database

Database of IKS Projects (3)

Eastern Cape [View subfolders] [View files (0)]
Freestate [View subfolders] [View files (0)]
Gauteng [View subfolders] [View files (2)]
KwaZulu-Natal [View subfolders] [View files (0)]
Limpopo [View subfolders] [View files (0)]
Mpumalanga [View subfolders] [View files (0)]
Northern Cape [View subfolders] [View files (0)]
North West [View subfolders] [View files (0)]
Western Cape [View subfolders] [View files (0)]

(c) 2004 IKS
top of page

Done

Illustration 4: IKS

4.2.3 IFLA-KM

4.2.3.1 ***Overview***

The IFLA-KM is the official site of the Standing Committee for Knowledge Management of the International Federation of Library Associations and Institutions, IFLA. It provides visitors with various repositories, including, amongst others, a collection of toolkits, a library of articles relating to knowledge management and a collection of activities organised by the committee. The site consists of a public section containing the five main repositories, SC Activities, Toolkits, Resources, People and Newsletters, and a members only section which has a discussion forum and an events calendar.

4.2.3.2 ***Client requirements***

As with the IKS project, the IFLA-KM project required the development and implementation of custom metadata schemas to effectively and accurately describe the various repositories' information objects. In addition to the custom metadata for the repositories, the project also required the addition of extra functionalities. One such addition is the calendar which was specifically developed and implemented for the IFLA-KM project. The newly added calendar had to accommodate different sets of calendar events, metadata for each event and the ability to browse these events.

4.2.3.3 ***Implementation and customisations***

The online discussion forum provides members with two threads to discuss knowledge management issues as well as general comments. An extra feature that was added for the IFLA-KM project is a graphical calendar which allows members to submit events that might be of interest to other members. The Calendar Module was built on top of the existing Statement Module, which greatly reduced the development time. This is a great example of how existing modules can be utilised to extend the features provided by the framework. Any user who is allowed access to the calendar can post new events in the public events calendar, whereas the private calendar is restricted to moderators. However, as with the document submission for the repositories, the events that are posted by members must first be accepted by the site's moderators. Users of the calendar can browse any of the available calendars and view each event's date, location and a description describing the purpose of the event.

As with the IKS project, the IFLA-KM project contains a list of people that are involved in the project. An advantage of using XML to describe a metadata schema is that that schema can be copied and seamlessly incorporated in another project. Refer to Table 11: Custom metadata fields for the Experts database. for more detailed information with regard to the people/experts metadata schema.

4.2.3.4 ***Lessons learned***

Very few lessons were learnt from the IFLA-KM project due to the fact that its requirements closely matched those of the IKS project. The following list details the knowledge that was gained from the implementation of the IFLA-KM project.

- *User requirements*

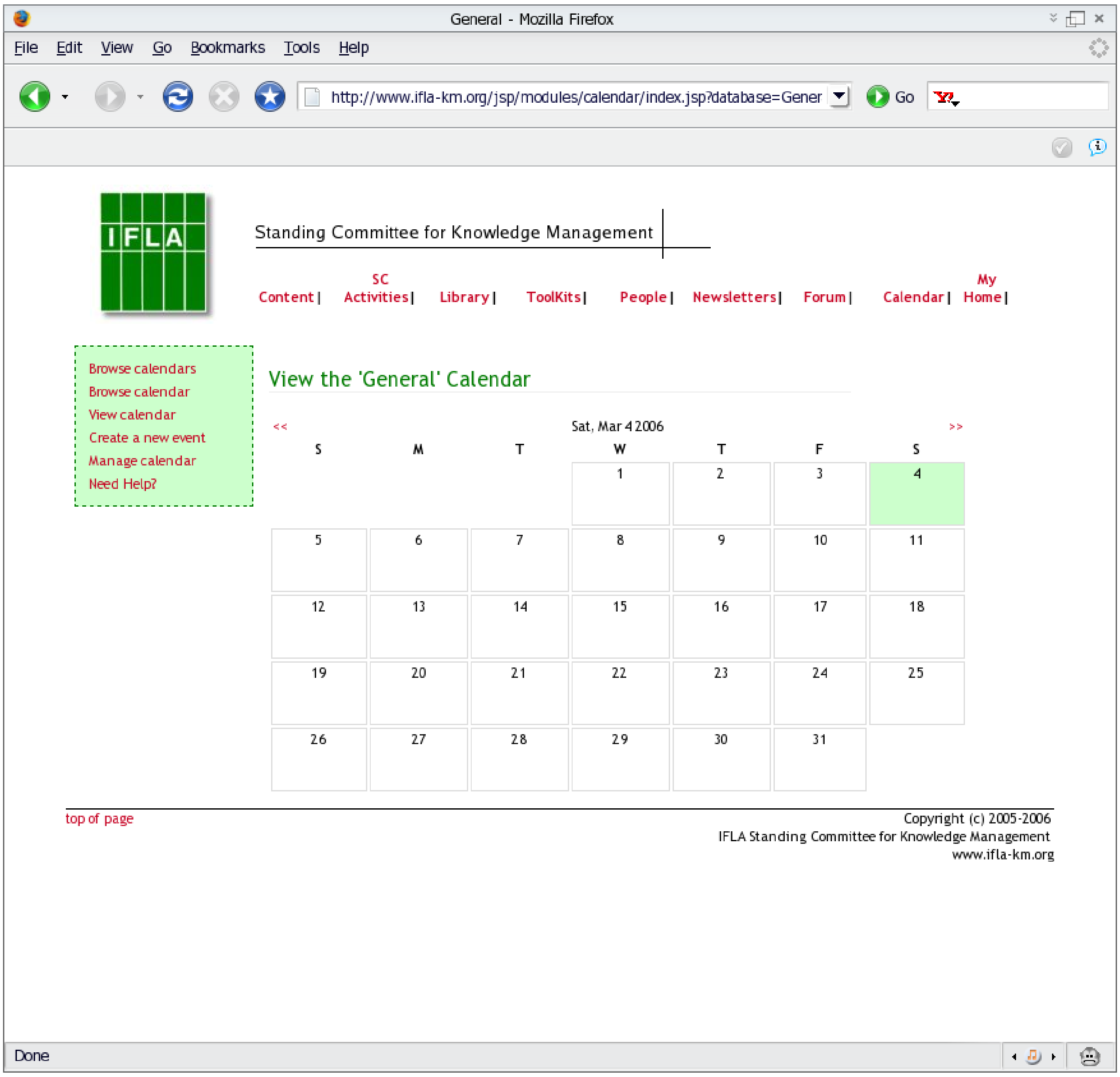
  The requirements, with respect to the metadata schemas, were better defined during the initial implementation phases in order to ensure that they remained unaltered for the remainder of the project. Moreover, the additional user requirements for the project's additional module were also defined in detail during the interviews. These strenuous guidelines were a direct result result from the lessons that were learnt during the IKS project.

- *Customisations*

  The lessons learnt from the IKS project resulted in a more streamlined implementation process of the various customisations for the IFLA-KM project. An example of such a customisation is the addition of the calendar module, for which the requirements and specification were set out before the development began. This resulted in less iterations being required to meet the functional and usability requirements of the project members.

- *Usability requirements*

  As with the IKS project, the usability testing of the IFLA-KM project consisted of a close working relationship with the project members. Very few changes had to be made to the framework, due to the fact that most of the usability issues which arose in the previous project were already addressed. The most significant usability testing of this project was concerned with the addition of the calendar module.


General - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://www.ifla-km.org/jsp/modules/calendar/index.jsp?database=Gener

Standing Committee for Knowledge Management

Content | SC Activities | Library | ToolKits | People | Newsletters | Forum | Calendar | My Home |

- Browse calendars
- Browse calendar
- View calendar
- Create a new event
- Manage calendar
- Need Help?

View the 'General' Calendar

<< Sat, Mar 4 2006 >>

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

top of page

Copyright (c) 2005-2006
IFLA Standing Committee for Knowledge Management
www.ifla-km.org

Done

Illustration 5: IFLA-KM

4.2.4 The Wellness Firm

4.2.4.1 ***Overview***

The Wellness Firm is a company that provides wellness services and solutions to corporate clients. The services range from complete physical assessments to online journalling with registered psychologists. With this implementation the framework was used to provide a corporate intranet for The Wellness Firm's employees. It consists of a set of repositories, an online discussion forum and an online calendar. Very little customisation was required for this implementation due to the fact that the root framework had by this time grew to incorporate the features that were added to the previous implementations. From this, it is evident that the project has reached maturity and that future implementations of the framework would also require very little customisations, if any.

4.2.4.2 ***Client requirements***

For the Wellness Firm's intranet, the client required the ability to organise documents, provide forums to its users and a calendar to store events. It was of high importance that The Wellness Firm implemented an intranet that was available to its employees due to the fact that they were based throughout the whole of South Africa. Each employee was therefore granted access to The Wellness Firm's various protocols, events calendar and online discussion forum. Moreover, the client required that the design be altered to match the corporate identity of The Wellness Firm to ensure that visitors to the site made aware of the link between the intranet and the firm.

This specific implementation was the first to incorporate the Workspace Module which provides additional functionalities to the digital library. These include the ability to manage different versions of the same document in the repository by allowing user to check-out a document, alter the content of that document, and the resubmitting the document as a new version of the original document. The various version of the document can then be accessed by a user when viewing the various documents contained within the workspace. This feature, combined with the use of the online discussion forum, proved invaluable to the client because of the fact that the contributors to the documents where scattered throughout the country, providing a true online collaborative workspace.

4.2.4.3 ***Implementation and customisations***

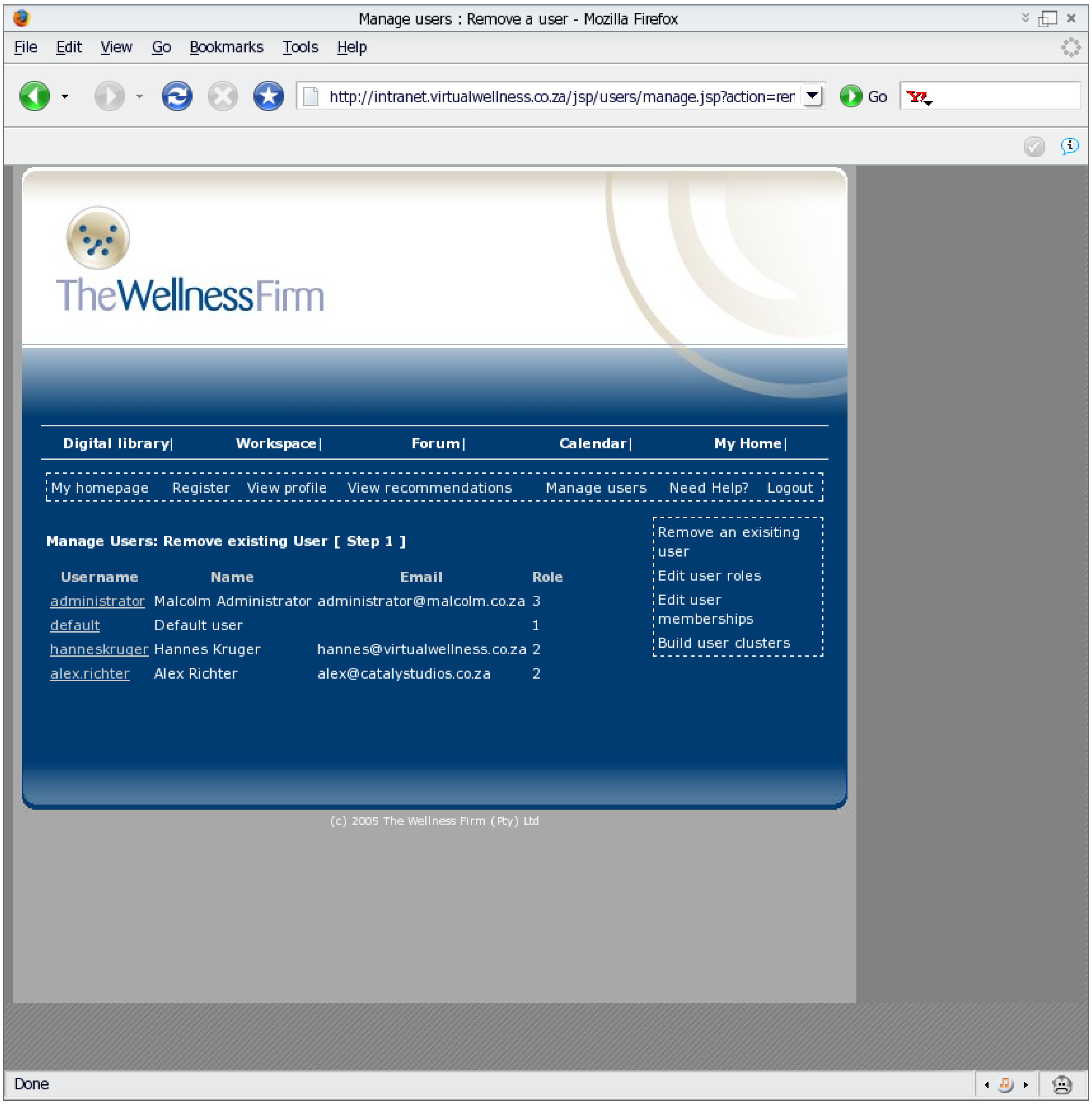
This implementation consists of two main repositories viz. Digital library and Workspaces , with plans to add an additional repository for People, an online discussion forum and a calendar. A key requirement of this project was to provide the employees and managers with an online collaborative workspace where they can download resources, publish documents and share thoughts in the discussion forum. Moreover, a public calendar is provided providing everyone with a centralised calendar where events and other time sensitive information can be published. The forum is divided into several main threads such as meetings, deadlines, resources and ideas. This once again provides employees and managers with a centralised point where information can be shared.

Another important aspect of the site's functionality is the management of users and their rights within the various repositories and forums. This is done by an administrator who can set their levels of access, for example User and Worker. All of the user administration is done through the member administration function provided by the core framework. This allows the administrator to set the memberships and roles for each of the members. Fortunately, the framework was designed with role based access in mind, thus providing the administrator the ability to specify the required roles for each action of each repository and forum thread.

4.2.4.4 ***Lessons learned***

The implementation of the framework for the Wellness Firm was the first instance where a commercial project was involved. This section will expand on the lessons that were learnt during the implementation phases.

- *User requirements*
  The user requirements, unlike the previous implementations, were defined much more clearly. This was because of the fact that the client had very specific needs for which a formal quote was issued. As a result, the user requirements did not change, unlike the other projects, which caused the implementation of the framework to follow a linear approach. The linear approach of the project was far removed from the time consuming iterative approach of the previous implements which resulted in the project being completed much quicker.
- *Customisations*
  Very few functional requirements were required for the implementation of this project. The vast majority of customisations were related to interface design and configuration of the various repositories' metadata schemas. This proved to be a great testament for the framework, which meant that if could be successfully be implemented in both research as well as commercial project without the need for many additional customisations.
- *Usability requirements*
  The usability requirements of this particular implementation were more subjective than those of the previous implementations. This was because of the client's own preferences with regard to interface design and functional layout. Unlike the research projects where the stakeholders had efficiency and effectiveness in mind, this project relied more on the visual impression of the site. This resulted in the usability testing being of lesser importance to the overall success of the project.


Manage users : Remove a user - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://intranet.virtualwellness.co.za/jsp/users/manage.jsp?action=rem
Go
TheWellnessFirm
Digital library| Workspace| Forum| Calendar| My Home|
My homepage Register View profile View recommendations Manage users Need Help? Logout
Remove an exisiting user
Edit user roles
Edit user memberships
Build user clusters
(c) 2005 The Wellness Firm (Pty) Ltd
Done

Manage Users: Remove existing User [Step 1]

Username	Name	Email	Role
administrator	Malcolm Administrator	administrator@malcolm.co.za	3
default	Default user		1
hanneskruger	Hannes Kruger	hannes@virtualwellness.co.za	2
alex.richter	Alex Richter	alex@catalystudios.co.za	2

Illustration 6: The Wellness Firm

4.3 Summary

This chapter discussed the various projects in which the framework has successfully been implemented and customised. Each of the projects were unique with regard to their target users and specifications. It was therefore necessary to first interview the stakeholders of each project in order to better understand the requirements. After obtaining the requirements and specifications, the framework could be implemented and customised to suit the needs of the project. This proved to be an excellent test of the framework's modularity and of its features to be customised. Another aspect that was addressed in this chapter was the framework's usability which was evaluated during the various implementation phases of each project.

It is evident from chapters 3 and 4 that because of the iterative development approach which was followed that the ADDIE model was crucial to the success of the framework. The ADDIE model consists of continuously revising past development and making changes as needed during the various development phases. In addition to revising the framework during the development stages, it was also necessary to revise the framework during the various implementations stages due to the new knowledge that was obtained.

The following chapter addresses the main research question as well as the various sub-problems that were identified during the beginning of this study and how the framework addressed each of these. Thereafter, the chapter reflects on what was learned during the literature survey, development and implementation of the framework. Lastly, the chapter ends off with future work which could further extend the features and usability of the framework that was developed during this study.

5 Chapter 5 – Conclusions and recommendations

5.1 *Introduction*

The preceding chapters first outlined the main objectives for the study and also the approach to be taken in order to first understand the research field by means of a literature survey. Following the literature survey, the next chapter describes how the knowledge obtained from researching scholar;y works and existing collaborative workspaces was used in order to develop and implement a framework that addresses the various user requirements. Lastly, the framework's ability to be customised and its usability is proven by reporting on the various research projects for which the framework has successfully been implemented.

This chapter answers the main research question of this study by stating the four sub-questions and how each was answered. Also included in this chapter is a summary of future work that can be done in order to further enhance the features and usability of the framework. The chapter concludes with a brief summary of the purpose of the research study as well as how it was executed.

5.2 *Summary findings regarding research objectives*

The purpose of this study was to determine the user requirements and design specifications of an online collaborative workspace. In order to address the main objective of the study, a further four sub-questions had to be answered. This section will describe how each of these four sub-questions were approached and subsequently answered.

5.2.1 What are the main components of current online collaborative workspaces as reported in the literature?

A literature review was done in order to determine the main components of current online collaborative workspaces. This was done to ensure that the study would address the needs of current researchers who need to collaborate on projects whilst being divided by location and time. The literature review comprised of first identifying existing systems and also research in the field of collaborative workspaces. A clear distinction had to made between existing collaborative systems and scholarly research due to the different approaches of each. In the case of existing systems, the best approach was to evaluate the online solutions that were available. This proved useful in determining the features and methods for performing tasks that are most commonly available to users.

A thorough understanding of the user requirements of an online collaborative workspace were obtained through the evaluation of the existing systems. Unlike the existing systems, the scholarly research did not provide online evaluation versions of the projects. In this case, the best approach was to do a thorough review of the publications that were available. A great advantage of the literature review was that the research provided a great insight in the ongoing research on the needs of users and how they can interact with one another through an online collaborative workspace. Thus, the existing systems provided a great insight into the features that are most commonly available to users, whereas the scholarly research provided an in-depth look into the motivation for the features that users need.

From the literature review and system evaluation, the following key components were identified:

- *Digital library*
  A digital library allows researchers to share both published and unpublished work on the Internet for review and participation by other researchers and scholars. The two main types of digital libraries that were identified, were a repository where published work could be shared between researchers and scholars alike and also a repository where unpublished work could be submitted for peer review. This allows researchers to quickly share their research findings with the research community, bridging the time associated with publishing one's findings in a publication such as a research journal. Moreover, scholars can search the repositories for research relating to their specific interests or needs.
- *Adaptive hypermedia*
  Adaptive hypermedia provides users with both customised content and presentation tailored to suit their individual needs. This is of great importance in systems that contain a diverse collection of research fields which might overwhelm the user. In the case of an online collaborative workspace, adaptive hypermedia could be used to limit the research fields to those in which the user has an active interest. This will allow the user to, for instance, quickly find the repositories containing information pertaining to his specific field of researcher. Another example is the customisation of the content's level of difficulty available to the user. For instance, a researcher would typically not be interested in reading an introduction relating to the fields of study in which he/she is involved, due to the fact that it would be of little use to him/her. It is therefore possible to present researchers with content of a higher level difficulty, whereas scholars would be

first be presented with introductions to the field of study. Moreover, the links available to the users could also be adapted to guide a user through his information gathering process.

- *Collaborative workspace*
  A collaborative workspace can be seen as the virtual place in which researchers, who are separated geographically, can share resources, information and ideas. Various forms of collaborative workspaces exist, each focussing on the specific needs of the project. One such example is the distinction between synchronous and asynchronous interaction between researchers. In the case of synchronous systems, participants would typically make use of a shared whiteboard and real-time chat in order to share ideas with one another. Asynchronous systems, in contras, provide researchers with a means to store information such as research findings and ideas persistently, allowing other participants access who might be in a different time zone. Another distinction could be made between ad hoc and formal systems where the focus is on how information is shared between researchers. Ad hoc systems would typically make use of email and other forms of sharing information without any formal structure for storing and describing the information. Formal systems, on the other hand, provides researchers with a highly structured method for describing, storing and accessing information.

5.2.2 What are the main design principles and functionalities of such a collaborative workspace and how can they be integrated in developing a modular open source framework?

A key design principle that was followed during the development of the framework was modularity. Modularity allows the framework to be easily adapted and customised to suit the requirements outlined for each implementation. Each of the components comprising the framework was developed with interoperability in mind. This allows, for instance, one module to re-use features exposed by other existing modules in the framework. By using this interoperability between the various modules, the development time is reduced and the duplicating of code is avoided. The list of modules that were developed include the following, a digital library, adaptive hypermedia engine, online discussion forum, statement database and user management.

Another key aspect in the development of the framework was the use of open source software. By using open source software, the development and implementation cost of the framework was greatly reduced. This reduction in cost makes the framework more accessible to researchers and their institutions. Another benefit of developing the framework using open source software, is that anyone can further extend and enhance the features that are available to suit their specific needs.

A certain level of progression, with regard to the design principles, was made during each implementation of the framework in the various projects. The lessons that were learnt during each implementation included certain limitations with respect to integration of modules' functions and also exposed a lack in naming conventions within both the database and public functions. These lessons provided insight into how certain aspects of the modularity of the framework could be altered in order to accommodate the modification and addition of modules. One such example of such progression is the refinement of the various module's configuration documents to streamline the implementation process. Another area where progression was made was with the interface improvements that were made for each of the functionalities exposed by the various modules.

5.2.3 To what extent can such a framework be customised for implementation in new or existing collaborative projects?

An important requirement of the framework was that it be customisable to suit the needs and requirements of various research projects' members. The customisation of the framework ranged from customising the interface to adding new features and modules. Before each implementation process, the needs of the research project was identified, and the ability of the current framework to meet these needs was assessed. A list of additional features and modules were identified if it became evident that the core framework could not satisfy all the requirements. These new features were then developed and integrated into the new implementation of the framework and would in some cases also be added to the core framework for future implementations.

In most cases, the framework's features were sufficient in meeting the research projects' needs, however, there were cases where some additional development was required. The additional features that were developed included a calendar containing various events, a map based search for the repository and management features such as moving documents between repositories. From this, it was evident that the initial user requirements that were identified during the initial interviews and literature surveys did not meet all the needs of today's research projects. However, the framework's ability to adapt and incorporate these new requirements was proven with each new implementation.

5.2.4 What are the usability requirements of such an online collaborative workspace and how should one go about to test the usability of such a framework?

Any system that is designed to be used by people need to conform to certain usability requirements. This is necessary to ensure that the system is learnable, memorable, effective, has a low low error rate and satisfies the user requirements. In order to determine in what manner a system adheres to these usability requirements, it is required that the system be tested by users. It was decided to subject the framework to various user tests during the development phases of the framework in order to ensure that any issues that might arise be addressed promptly. The usability testing consisted of working closely with the project experts who provided constant feedback with respect to the usability requirements. After each feedback session, the various remarks and input could be analysed and the appropriate alterations to the framework could be made. In the end, the finalised framework succeeded in meeting the usability requirements of the users by allowing them to easily learn the system and effectively perform their required tasks.

One implication of the expert testing that was identified during the various implementations of the framework was that it did not provide an accurate view of the usability of framework. During the implementation of the framework in the various projects, it became evident that users who were less proficient in computers had some difficulties when performing routine tasks. This resulted in certain functionalities being revisited in order to accommodate the feedback that was provided by the users of the system.

5.2.5 What are the user requirements and design specifications of an online collaborative workspace developed in open source software?

The main research question of this study can be divided into three parts, viz: user requirements, design specifications and the use of open source software. This section will expand on how each of these parts were addressed.

5.2.5.1 ***User requirements***

The main objective of the research study is to develop a framework that meets certain user requirements. A literature survey was performed in order to establish what features are most commonly available in existing collaborative workspaces and those were implemented as follows:

- *Digital library*
  The digital library allows researchers the ability to submit published or unpublished documents to various repositories. Each document is accompanied by a Dublin Core as well as customised metadata schemas that describe properties including title, author, date and location. Users of the digital library can either browse the directory hierarchy of a repository for documents, or can perform a search to locate a specific document. The digital library module was developed to support two types of repositories, viz.
  - A standard repository
    A key characteristic of the standard repository is that documents that are submitted to the library are finalised research works. The standard repository's functionalities were modelled on that of Dspace [Dspace 2006], Greenstone [Greenstone 2006] and ePrints [Eprints 2005] which were reviewed in chapter 2.
  - Workspace repository
    Unlike the standard repository, the workspace allows researchers to submit draft versions of their findings to the digital library for peer review. Any document in the workspace can therefore be checked-out by other users and edited or revised and then resubmitted to the library. The emphasis of the workspace is therefore to allow researchers a common repository where collaboration assists them in their research efforts.
- *Adaptive hypermedia*
  In addition to providing static content, the framework also supports adaptive hypermedia. The adaptive hypermedia engine provides various methods in which the content, interface and links can be customised for each user of the site. A key component of the adaptive hypermedia engine is the profiling of users, which allows the system to store user preferences and navigational behaviour. This information is then used when a user visits a page to determine exactly how the various components can be customised. The customisation can be grouped into the following three groups, viz.
  - Content
    The content of the page can be adapted using, for instance, stretch text which will hide information from the user which the system deems to be of less importance. In addition to stretch text, the system uses the user's perceived knowledge of the subject matter to customise the level of difficulty of the

content that is available.

- Links
  Link level adaptation assists the user in navigating the information space by ordering, hiding and emphasising links on a page. Key to the success of link adaptation is the ability to know both where a user has been and also how the information is to be accessed by the user. The content of the framework is structured, using both Learning Object and customised metadata schemas, which allows the system to determine the best order in which to guide the user through the various pages.

- Recommendations to users
  An additional benefit of using metadata to describe documents and creating user profiles is that the system can use this information to provide recommendations to users. This is done by comparing users' profiles with one another to recommend resources that were accessed by groups of similar users. Moreover, a user's personal profile can be compared to that of the documents in the framework to recommended resources that match a user's personal preferences.

◆ *Online discussion forum*
The online discussion forum allows researchers to enter into a discussion regarding specific topics. Each topic is represented by a room in the forum which exists out of a threaded discussion allowing users to both post new or reply to existing comments. Users can browse the collection of discussion forums or they can search the rooms to find answers to specific questions.

◆ *Statement database*
Another simple, yet effective, method for conveying ideas or comments is the statement database. A statement can range from a one line comment made by a user or a quote that can inspire new research ideas. Each statement is accompanied by metadata which described properties including the original source of the statement or quote, additional comments and also a URL providing more detailed information.

◆ *User management*
The framework was designed to be accessed by various types of users each having his own privileges. This is accomplished by a user management component that allows an administrator to add, edit and remove users. Each user is assigned a specific role within the framework which determines which actions

he/she can perform in each of the modules. For example, a worker user can submit documents to a repository, however, only a moderator can accept those documents so that they become visible to the public.

5.2.5.2 ***Design specifications***

The ability of the framework to be further enhanced by users played a vital part in the design specifications. This was achieved by structuring the framework into various modules which interact with one another. It is therefore possible to add or modify existing modules to further extend the features and functionalities of the framework.

Each of the main components of the framework is represented by its own module which is responsible for the configuration and implementation of its features. The base class, which is implemented by each of the modules, provides various base functionalities that maximises code re-use whilst also allowing for easy integration. The six main modules, which are described in detail in chapter 3, are:

- *Graphical User Interface*
  This module is responsible for rendering the content and links which are generated by the various modules in the framework. An example of the integration between the GUI and Users module is the ability to specify a specific interface in a user's profile which will then be used by the GUI to render content and links.
- *Users*
  The users module is used to manage the collection of users as well as their respective profiles. An administrator has various management options to, for instance, register or edit members and also assign rights to users. User authentication and authorisation is handled by the users module and is used by the various modules to ensure security throughout the framework.
- *Repository*
  The repository provides the framework with a digital library which allows users to submit new documents to various repositories and also access existing documents by using either the browsing or searching functionalities. Key to the success of the repository is the use of its metadata classes which is also exposed to the other modules allowing efficient exchange of information.
- *Content*
  The content module provides the adaptive hypermedia capabilities of the framework by allowing users to be presented by customised content and

navigation. This is done by using the user's profile, which is exposed by the users module, in conjunction with the metadata, describing the content, to determine what information should be displayed and how links should be altered.

- *Forum*
  The online discussion forum provides users with a threaded discussion board. Each of the topics are described using metadata which can be queried by the user module to determine whether a specific room would be of interest to a particular user. This is another example of how the interaction between the various modules' components allows for easy integration of features.
- *Statement*
  A statement is a single comment or quote which was made by a user of the framework or any other person. Each statement is described using the metadata classes which are exposed by the repository module. There also exists various statement databases, each described by metadata, which can be queried by the user module to determine whether it would be of interest to a particular user.

Another important component of the design is the methods for storing information used throughout the framework. This is done by using a combination of XML, SQL and serialised Java objects. Each module's configuration is stored in XML, allowing both humans and the framework to make use of it. The relational database allows for searching of information, such as document metadata. Lastly, serialised Java objects are used to minimise the amount of time required to process large XML documents, which are stored internally in an object and serialised for future use.

5.2.5.3 ***Open source software***

The last part of the research question deals with the use of open source software to develop the framework. This was achieved by using a combination of open source solutions which are freely available. The various technologies that were utilised include:

- *MySQL*
  A relational database is used as the primary storage engine for the metadata describing the various information objects contained within the framework. MySQL [MySQL 2006] was chosen as the database server because of its scalability, flexibility, speed and ease of use. The framework, can however, be configured to use any relational database server that conforms to the SQL standards.
- *Perl*

The installation of the framework consists of various Perl [perl.com 2006], [perl.org 2006] scripts that are used to copy configuration and source files for the new implementation. In addition to copying files, the scripts are also used to change settings in the configuration files to allow the various modules to make use of the new implementation's namespaces and directory structures.

- *Java*
  The primary programming language that was used throughout the framework is Java [Sun 2006]. Java was used because of its ability to work seamlessly on various platforms such as Linux, Apple Macintosh and Microsoft Windows. The framework's implementation possibilities and accessibility is greatly enhanced by using a language that is so widely supported.
- *Jakarta Tomcat*
  Jakarta Tomcat [Jakarta 2006]is a web server used to server the Java Server Pages used throughout the framework. It was developed by the Apache Software Foundation [Apache 2006] and is continuously updated to ensure better functionality and performance.
- *XML and XSL*
  Each module's configuration as well as the content of the framework is stored as XML [xml.org 2006] [xml.com 2006] documents. XML allows for easy configuration because of its ability to be customised for each situation and the fact that it's understandable by both humans and the framework. Moreover, XML is used to to represent the structure of the modules' output which is rendered as HTML using XSL.

5.3 ***Reflections***

The iterative development model that was followed during the development and implementations of the framework was the ADDIE model. It consisted of constantly revising past efforts and reflecting on what needs to be changed to ensure a quality framework. Following is a description of the iterative process that was followed, first during the development stages, and then during the implementation phases.

- *Development stages*
  As stated previously, the aim of the framework was for it to be modular, scalable, flexible and customisable. It was therefore necessary to constantly revisit past modules to determine whether they could be optimised for better integration within the rest of the framework. An example of such revisit is the Metadata class

that was initially developed for the Repositories module. During the development of the Content module, it became evident that metadata had to be used to describe the various pages contained in the adaptive hypermedia engine. It was therefore necessary to revisit the Metadata class and Repository module to determine how these could be changed to allow the Content module to integrate seamlessly with the Metadata class.

- *Implementation phases*
  It was also required to revisit the framework during the various implementation phases. This was because of the fact that each implementation required certain customisations, such as additional features and custom metadata schemas. It was therefore necessary to determine which existing modules could be used and also how each could be customised to provide the new functionalities required by the project. Unlike the development process, this was more interactive with constant feedback from the project stakeholders in the form of functional as well as usability changes that needed to be made. The feedback that was received after each iteration was reviewed and the appropriate changes were made to framework. In certain cases, the new functionalities and modules were added to the original framework.

5.3.1 Methodological reflection

The approach that was followed during the development of the framework as to first determine a broad set of features and then developing a prototype which implemented those requirements. During the various development stages, the details of each feature were better defined, allowing for the framework to rapidly and constantly evolve. This approach had certain advantages and disadvantages, viz.

- *Disadvantages*
  The development process could not follow conventional methodologies, which had the result that there were never clear objectives because of the constantly changing user requirements. This resulted in modules being revised constantly to include new features and integration, whilst still being compatible with existing modules.

- *Advantages*
  The addition and removal of functionalities were, at times, a simple process and never involved complex decision making or motivation processes. This resulted in the framework's constant evolution ensuring that it conforms to all of the user's

constantly changing needs. The modular architecture of the framework allowed for easy customisation and integration of the various components. Also, the use of XML for the configuration means that the alteration of metadata schemas is easy which resulted in the compliance with each projects information needs. Lastly, the three-tiered architecture allowed for the separation of business and interface logic, which resulted in personalised presentation for individual users as well as the customisation of the interface to suit a project's specific needs.

5.3.2 Substantive reflection

This section will briefly discuss how the framework that was developed was influenced by the literature survey and its findings. The user requirements called for an online collaborative workspace to be developed for researchers and institutions. From the literature survey, the various components required to provide such a framework were deduced. These components, however, do not exists in an integrated framework providing all of the functionalities required by an online collaborative workspace. The motivation, therefore, for the development of this project was to provide a single framework which integrated all of the components that were identified during the literature survey. Moreover, attention was given to the customisation of the framework to suit the needs of future projects. Lastly, the framework was developed using open source software which was identified during the literature survey, to ensure accessibility to researchers and institutions with limited financial resources.

5.3.3 Scientific reflection

Any research study aims to benefit the research community by providing insights into lessons that were learnt as well as recommendations. This study was performed to determine the user requirements and design specifications of an online collaborative workspace built on open source software. In the process of developing the framework, the following four benefits, with regard to development strategies, were identified:

- *Modularity and integration*
  The success of the framework in the various instances where it has been implemented proves that it is possible to integrate the various components that were identified during the literature survey. Also, the modular architecture of the framework proved to be the best model to follow due to the success of the customisation of each implementation with respect to its functionalities. Lastly, the integration of the various modules' classes, such as the Repository module's Metadata class, is a testament to the benefits that can be reaped from effective

code reuse.

- *Three-tiered architecture*
  The three-tiered architecture that was used to build the system ensured that the customisation or removal of any of the components did not disrupt the overall framework's ability to function properly. This is because of the separation of the database, processing and presentation layers of the framework. It was therefore possible to make changes to, for instance, the interface without the need to alter the underlying programming and database. This greatly enhanced the framework's ability to be customised as well as reduced the amount of time and effort needed for each such customisation.
- *Iterative development approach*
  Another key to the success of the development and implementations of the framework was the use of the ADDIE model to constantly reflect and revisit past efforts during the various phases. One such an example is the constant re-evaluation of the modules and their respective classes and how they could be integrated to optimise code re-use and modularity. Moreover, the interactive and iterative approach that was followed during the various implementations ensured that the user requirements with respect to functionalities and usability were met.
- *Open source software*
  The various programming languages, database servers, mark-up languages and server software available were sufficient in the development of the framework. This proves that the use of open source software has become a reality in the development of sophisticated projects. It is therefore possible for institutions with limited resources to obtain and develop frameworks that meet their needs.

5.4 ***Recommendations***

This section will provide information with regard to the policies and practice that were followed during the development of this framework and how they could benefit similar projects. Moreover, a list of future work that can further expand the framework is provided.

5.4.1 Policy and practice

Certain policies and practices were followed during the development of this framework. This section will expand on the iterative development process, integration of functionalities, modular design and interactive development of the framework.

- *Integration of functionalities*

  This research project first set out to determine which components are required in an online collaborative workspace. After obtaining a better understanding of the various components needed to provide such a system, the next step was to integrate the various components into a single framework. The motivation behind integrating the various components into a single framework, was two-fold, namely to determine whether such a system would benefit researchers and institutions as well as whether development efforts could be minimised. By working closely with researchers and performing usability resting during the development phases it was determined that the integration of components proved to benefit users in their research efforts. Moreover, with respect to development, the benefits of integration became evident due to the successful integration of the various components' classes, reducing development time and increasing interoperability. One should therefore aim to integrate functionalities in a framework where the opportunity exists that each component could benefit from features that are provided by the rest of the system.

- *Modularity*

  The framework was intended to be implemented in various research projects, each with their own requirements. It was therefore decided from the start that a modular approach was to be followed with respect to the development of the framework. The framework comprises of both functional and supporting core modules. Before development began, the various modules for each group were identified and also how each module could be integrated to form a complete framework. By making use of a modular framework, it was easy to add, modify and remove modules to suit each implementing project's individual needs. Another benefit of using a modular approach, was that certain functionalities provided by the various modules' classes could be shared. By sharing these common functionalities, the development time could be reduced whilst code reuse was improved. For these reasons, any project which aims at providing customisable features to its users should therefore aim at separating its components into various modules to allow for the addition and modification of modules if the need arises.

- *Iterative development*

  During the development stages, the specifications were continuously revised due to ever changing user requirements. The iterative development of the project consisted of constantly revisiting past work and revising components in order to ensure that all user requirements were met. Another example where an iterative approach proved to be beneficial was with the various implementations of the framework. Each implementation required that the current version of the framework had to be revisited in order to determine which modules and components could be adapted to meet the new user requirements. The knowledge that was obtained after each implementation could then again be used to further enhance the functionalities provided by the original framework. Developers and project stakeholders should therefore follow an iterative development approach in projects where the user requirements constantly change and also in cases where a core framework is to be implemented in various projects with specific needs.

- *Interactive development*
  The development specifications of the project were constantly revised in order to suit the ever changing user requirements. Close interaction with project stakeholders is crucial in such an ever evolving project in order to ensure that the user requirements were always understood and implemented. In addition to meeting user requirements, another benefit of interactive development is that the usability of the framework can also be continuously assessed ensuring the efficiency, effectiveness and user satisfaction. One should therefore aim to work closely with the various stakeholders in order to ensure that user and usability requirements are met in a timely manner.

5.4.2 Future work

The current core framework was developed based on the requirements of the original projects as well as the specific customisations that were required by the various implementations. From the literature survey, however, it can be seen that there are still features that could be implemented to further enhance the features and usability of the framework. Following is a list of four areas where future work on the framework could focus.

- *Synchronous communication*
  Currently the framework only supports asynchronous communication through, for instance, digital libraries and discussion forums. This was sufficient for the projects for which the framework was developed. However, it would be of great benefit to future projects to also implement a collection of synchronous communication channels. These channels could include an online chat application, shared whiteboard and video conferencing functionalities.

- *Installation module*
  The complexity of the framework's installation scripts make it very difficult for ordinary users to easily implement a new instance. In order to make the framework more accessible to ordinary users, a simple to use graphical installation application could be developed to ease the installation process. Ideally, this installation application would provide users with a list of module which should be implemented, wizards to edit the configuration documents and also a means to integrate customised modules in to the framework.
- *Editor for adaptive content*
  The content for the adaptive hypermedia is stored in various XML documents using a structure that defines the metadata, sections and links of each page. These XML documents are complex to create and maintain manually. An editor for the various pages within the framework would therefore greatly assist moderators in creating content that could be adapted to meet each user's personal profile. Such an editor would have an HTML editor which could be used to create the actual content and then various wizards to create, for instance, sections with adaptive properties such as stretch text.
- *Plug and play metadata schemas*
  The framework's repositories support the use of both standard metadata schemas, such as Dublin Core, and also custom schemas developed specifically for the repository's subject matter. These metadata schemas are created manually for each repository and stored in XML documents. A plug and play metadata editor or wizard could be developed in order to assist users in the creation of these metadata schemas. Such a wizard could provide users with a list of existing metadata schemas from which a selection could be made, and then also include an editor for the creation of new metadata schema.

5.5 ***Final conclusions***

In conclusion, many collaborative workspace that are currently used by researcher institutions and individuals. From the literature survey it became evident that there is in deed a lack of systems that meet all the user requirements of a truly online collaborative workspace. This research study's purpose was therefore three-fold. Firstly, all the user requirements of an online collaborative workspace were identified by performing both a literature survey as well as informal interviews with research experts. After establishing all the user requirements, the next step was to develop and implement a framework that would meet all the requirements and also allow for future enhancements. The final framework included modules supporting both static and adaptive content, a digital library, online discussion forum, statement database and user management. In addition to developing such a framework, another requirement was to do so using open source software, making the framework accessible to research institutions with limited resources.

The flexibility, scalability and conformance to user requirements of the framework were proven by successfully implementing it in various research projects as well as commercial web sites.

Bibliography

Anderson, William L.. 1997, Digital libraries: A brief introduction, SIGGROUP, vol. 18, no. 2, pp.5-6
Bailey, Hall, Millard, Weal, 2002, Towards Open Adaptive Hypermedia, In Proceedings of the Second International Conference on Adaptive Hypermedia and Adaptive Web Based Systems, Spain, Springer, pp. 36-46
Bao, Y., Zou, H., Zhang, J. 2005, Using PACT in an e-commerce recommendation system, Proceedings of the 7th international conference on Electronic commerce, Xi'an, China, ACM Press, pp. 466-470
Bobulous , 2004, Why you should use open source software, 2004. [Online] Available at: http://www.bobulous.org.uk/misc/openVclosed.html
Boll, Susanne, 2003, MM4U - A framework for creating personalized multimedia content
Branschofskyh, M., Chudnov, D., 2002, DSpace: Durable Digital Documents, JCDL 2002, Portland, Oregon, USA, ACM Press, pp. 372-373
Brody, Hajjem, Harnad, 2005, The Research-Impact Cycle
Brusilovsky, et al. 1998, Methods and Techniques of Adaptive Hypermedia, Adaptive Hypertext and Hypermedia, vol. 6, no. 2, pp.1-43
Brusilovsky, P., Schwartz E., Weber, G. 1996, ELM-ART: An Intelligent Tutoring System on World Wide Web, Proceedings of the Third International Conference on Intelligent Tutoring Systems, London, UK, Springer-Verlag, pp. 261-269
Explore Open Source Alternatives, 2004. [Online] Available at: http://cpr.ca.gov/report/cprrpt/issrec/stops/it/so10.htm
Carr, et al.. 2001, Conceptual linking: ontology-based open hypermedia, Proceedings of the 10th international conference on World Wide Web, , ACM Press, pp. 334-342
Carr, L., Hall, W., De Roure, D.. 1999, The evolution of hypertext link services, ACM Computing Surveys (CSUR), vol. 31, no. 4, pp.1-6
Budapest Open Access Initiative, 2002. [Online] Available at: http://www.soros.org/openaccess/read.shtml
Cheverst K., et al. 2000, Using Context as a Crystal Ball: Rewards and Pitfalls, Proceedings of Workshop on 'Situated Interaction in Ubiquitous Computing', vol. 7, no. 3, pp.17-26
Cheverst, K., Mitchell, K., Davies, N. 2002, The role of adaptive hypermedia in a context-aware tourist GUIDE, Communications of the ACM, vol. 45, no. 5, pp.47-51
De Bra, P., et al.. 2003, AHA! The Adaptive Hypermedia Architecture, Proceedings of HT'03, , ACM Press, pp. 81-85
De Bra, P., Houben, G., Wu, H. 1999, AHAM: a Dexter-based reference model for adaptive hypermedia, Proceedings of the tenth ACM Conference on Hypertext and hypermedia : returning to our diverse roots: returning to our diverse roots, Darmstadt, Germany, ACM Press, pp. 147-156
Debevc, M., Rajko S., Donlagic, D. 1994, Adaptive bar implementation and ergonomicst, Informatica : Journal of Computing and Informatics, vol. 18, no. 3, pp.357-366
DeRose, S.J.. 1989, Expanding the Notion of Links, Proceedings of ACM Hypertext '89, Pittsburgh,

PA, ACM Press, pp. 249-257
Dommel, H. P. 2005, The challenges of ambient collaboration, Proceedings of the 2005 conference on Diversity in computing, New Mexico, USA, ACM Press, pp. 10-13
Dourish, P., Belotti, V. 1992, Awareness and coordination in shared workspaces, Proceedings of the 1992 ACM conference on Computer-supported cooperative work, Ontario, Canada, ACM Press, pp. 107-114
Geyer W., et al. 2003, Supporting activity-centric collaboration through peer-to-peer shared objects, Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work, Florida, USA, ACM Press, pp. 115-124
Advantages of open source software, 2000. [Online] Available at: http://eu.conecta.it/paper/Advantages_open_source_soft.html
Gutteridge, C (2005). New Developments in EPrints. In: Proceedings of CERN workshop on Innovations in Scholarly Communication (pp.). :
Hajjem, Hanrad, Gingras, 2005, Ten-Year Cross-Disciplinary Comparison of the Growth of Open Access and How it Increases Research Citation Impact
Halasz, F., Schwartz, M. 1990, The Dexter Reference Model, Proceedings of the NIST Hypertext Standardization Workshop, vol. , no. , pp.95-133
Halasz, F., Schwartz, M.. 1994, The Dexter hypertext reference model, Communications of the ACM, vol. 37, no. 2, pp.30-39
Hall, W., Davis, H., Hutchings, G.. 1996, Rethinking Hypermedia: The Microcosm Approach. Massachusetts, Kluwer Academic Publishers
Han, Eui-Hong, Karrypis, George. 2005, Feature-based recommendation system, Proceedings of the 14th ACM international conference on Information and knowledge management, Germany, ACM Press, pp. 446-452
Herlocker, J., Konstan, J., Riedl, J. 2000, Explaining collaborative filtering recommendations, Proceedings of the 2000 ACM conference on Computer supported cooperative work, Pennsylvanie, USA, ACM Press, pp. 241-250
James Y.L. Thong, Weiyin Hong, Kar Yan Tam. 2004, What leads to acceptance of digital libraries?, Communications of the ACM, vol. 47, no. 11, pp.78-83
Kobsa, Koenemann, Pohl. 2001, Personalised hypermedia presentation techniquesfor improving online customer relationships, , vol. 16, no. 2, pp.111-155
Kristofic, A., Bielikova, M. 2005, Improving adaptation in web-based educational hypermedia by means of knowledge discovery, Proceedings of the sixteenth ACM conference on Hypertext and hypermedia, Austria, ACM Press, pp. 184-192
The Open Archives Initiative Protocol for Metadata Harvesting, 2004. [Online] Available at: http://www.openarchives.org/OAI/openarchivesprotocol.html
Why Digital Libraries, 1995. [Online] Available at: http://www.ukoln.ac.uk/services/papers/follett/lesk/paper.html
Levy, David M.. 1995, Going digital: A look at assumptions underlying Digital Libraries, Communicatins of the ACM, vol. 38, no. 4, pp.77-85

Miles-Board, T. Everything Integrated: A Framework for Associative Writing in the Web, 2004
Mobasher, et al. 2001, Effective personalization based on association rule discovery from web usage data, Proceedings of the 3rd international workshop on Web information and data management, Georgia, USA, ACM Press, pp. 9-15
Muller, M. J., et al. 2004, One-hundred days in an activity-centric collaboration environment based on shared objects, Proceedings of the SIGCHI conference on Human factors in computing systems, Vienna, Austria, ACM Press, pp. 375-382
Nielsen, Jacob. 1993, Usability Engineering. Massachusetts, USA, Academic Press
The Cathedral and the Bazaar, 2002. [Online] Available at: http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/index.html
A Software Engineering approach to Libre Software, 2004. [Online] Available at: http://www.opensourcejahrbuch.de/2004/pdfs/III-3-Robles.pdf
Schafer J., Konstan, J., Riedl, J, 2002, Meta-recommendation systems: user-controlled integration of diverse recommendations, Proceedings of the eleventh international conference on Information and knowledge management, Virginia, USA, ACM Press, pp. 43-51
Shneiderman, B., Plaisant, C.. 2005, Designing the user interface. USA, Pearson Education
Shreeves, Kirkham, Kaczmarek, Cole. 2003, Utility of an OAI Service Provider Search Portale, Proceedings of the 3rd ACM/IEEE-CS joint conference on Digital libraries, , ACM, IEEE, pp. 306-308
Spellman, P., et al, 1997, Collaborative virtual workspace, Proceedings of the international ACM SIGGROUP conference on Supporting group work: the integration challenge, Arizona, USA, ACM Press, pp. 197-203
Strydom, H., Fouche, C.B., Delport, C.S.L.. 2002, Research at grass roots. , Van Schaik Publishers
Suchman, L. 1987, . , Cambridge University Press
Tang, J. C., Leifer, J. L. 1988, A framework for understanding the workspace activity of design teams, Proceedings of the 1988 ACM conference on Computer-supported cooperative work, Oregon, USA, ACM Press, pp. 244-249
Tansley, et. al.. 2003, The DSpace Institutional Digital Repository System: Current Functionality, , vol. , no. , pp.87-98
van den Akker, J. 1999, Design methodology and developmental research in education and training. The Netherlands, Kluwer Academic Publishers
Wiederhold, Gio. 1995, Digital libraries, Value and Productivity, Communications of the ACM, vol. 38, no. 4, pp.85-97
Wiederhold, Gio. 1995, Digital libraries, Value and Productivity, Communications of the ACM, vol. , no. , pp.85-97
Wilkinson, R., Lu, S., et al. 2000, Generating Personal Travel Guides from Discourse Plans, Proceedings of the International Conference on Adaptive Hypermedia and Adaptive Web-based Systems, Italy, , pp.
Witten, McNad, Boddie, Bainbridge, 2000, Greenstone: A Comprehensive Open-SourceDigital Library Software System, Proceedings of the fifth ACM conference on Digital libraries, , ACM Press, pp. 113-121

Web Developer's Virtual Library, 2006. [Online] Available at: http://www.wdvl.com
JavaScript.com (TM), 2006. [Online] Available at: http://www.javascript.com
XML From the Inside Out -- XML development, XML resources, XML specifications, 2006. [Online] Available at: http://www.xml.com
XML.org, 2006. [Online] Available at: http://www.xml.org
Java Technology, 2006. [Online] Available at: http://java.sun.com
The Comprehensive Perl Archive Network, 2006. [Online] Available at: http://www.cpan.org
The Perl Directory, 2006. [Online] Available at: http://www.perl.org
The Source for Perl -- perl development, perl conferences, 2006. [Online] Available at: http://www.perl.com
Three Tier Software Architectures, 2006. [Online] Available at: http://www.sei.cmu.edu/str/descriptions/threetier_body.html
World Wide Web Consortium, 2006. [Online] Available at: http://www.w3.org
Sun Microsystems, 2006. [Online] Available at: www.sun.com
IBM, 2006. [Online] Available at: www.ibm.com
Bonsai Project Home Page, 2006. [Online] Available at: http://www.mozilla.org/projects/bonsai/
Thunderbird - Reclaim your inbox, 2006. [Online] Available at: http://www.mozilla.org/products/thunderbird
Firefox - Rediscover the Web, 2006. [Online] Available at: http://www.mozilla.org/products/firefox
Slackware Linux, 2006. [Online] Available at: www.slackware.com
Debian Linux, 2006. [Online] Available at: www.debian.org
Red Hat Linux, 2006. [Online] Available at: www.redhat.com
Jakarta, 2006. [Online] Available at: jakarta.apache.org
Xerces, 2006. [Online] Available at: xml.apache.org
Mod_Perl, 2006. [Online] Available at: perl.apache.org
The Apache HTTP Server Project, 2006. [Online] Available at: http://httpd.apache.org
Home of the Mozilla Project2006
MySQL AB :: The world's most popular open source database, 2006. [Online] Available at: http://www.mysql.com
The Linux Kernel Archives, 2006. [Online] Available at: http://www.kernel.org
Linux.org, 2006. [Online] Available at: http://www.linux.org
The Apache Software Foundation, 2006. [Online] Available at: http://www.apache.org
Gna!: Welcome, 2006. [Online] Available at: http://www.gna.org
Welcome to freshmeat.net, 2006. [Online] Available at: http://freshmeat.net
GNATS - GNU Project, 2006. [Online] Available at: http://www.gnu.org/software/gnats/
Home :: Bugzilla, 2006. [Online] Available at: http://www.bugzilla.org
Tinderbox, 2006. [Online] Available at: http://www.mozilla.org/tinderbox.html
CVS - Concurrent Versions System, 2006. [Online] Available at: http://www.nongnu.org/cvs/
SourceForge.net, 2006. [Online] Available at: http://www.sourceforge.net
IT Facts, 2004. [Online] Available at: http://blogs.zdnet.com/ITFacts/wp-

mobile.php?p=5906&more=1
Open Source - GNU Project - Free Software Foundation (FSF), 2006. [Online] Available at: www.gnu.org/philosophy/free-software-for-freedom.html
A Brief History of Free/Open Source Software Movement, 2000. [Online] Available at: http://www.openknowledge.org/writing/open-source/scb/brief-open-source-history.html
Open Source Initiative (OSI), 2006. [Online] Available at: www.osi.org
phpgroupware.org, 2006. [Online] Available at: http://www.phpgroupware.org
GUIDE: Introduction, 2006. [Online] Available at: www.guide.lancs.ac.uk/overview.html
Introducing DSpace: DSpace Federation, 2006. [Online] Available at: www.dpsace.org
Greenstone Digital Library Project, 2006. [Online] Available at: http://www.greenstone.org
WikiPedia, 2006. [Online] Available at: http://en.wikipedia.org
EPrints: Supporting open access, 2005. [Online] Available at: http://www.eprints.org
, 1998. [Online] Available at: http://www.dlib.org/metrics/public/papers/dig-lib-scope.html

Appendices

Appendix A

Content Configuration XML

```
    <subjects>


        <!--

            List of subjects describing the Content of the site.

            e.g.

                <subject id="$id" name="$name" role="DEFAULT|USER|WORKER|ADMIN|SUPER">

                    <description>$description</description>

                    <malcolm:themes>

                        <theme name='$name' value='$float_value' />

                    </malcolm:themes>


                    <security>

                        <action name="$action_name" role="DEFAULT|USER|WORKER|ADMIN|SUPER" />

                    </security>

                </subject>


                !!!!NB: add a new entry to the dc:subject meta data field in */config/extra/metadata/fields/field[name =
'subject']


        -->


        <subject id="AH" name="Adaptive hypermedia" role="USER">

            <description></description>

            <malcolm:themes>

                <theme name='AH' value='100.0' />

                <theme name='AI' value='75.0' />

                <theme name='Datamining' value='60.0' />

            </malcolm:themes>


            <security>
                <action name="browse" role="USER" />
                <action name="read_article" role="USER" />
                <action name="submit" role="ADMIN" />
                <action name="edit_themes" role="ADMIN" />
                <action name="edit_requirements" role="ADMIN" />
                <action name="edit_links" role="ADMIN" />
                <action name="remove_article" role="ADMIN" />
            </security>

        </subject>

        <subject id="HCI" name="Human Computer Interaction" role="USER">
            <description></description>
            <malcolm:themes>
                <theme name='HCI' value='100.0' />
                <theme name='AH' value='50.0' />
                <theme name='AI' value='50.0' />
            </malcolm:themes>

            <security>
                <action name="browse" role="USER" />
                <action name="read_article" role="USER" />
                <action name="submit" role="ADMIN" />
                <action name="edit_themes" role="ADMIN" />
                <action name="edit_requirements" role="ADMIN" />
                <action name="edit_links" role="ADMIN" />
                <action name="remove_article" role="ADMIN" />
            </security>

        </subject>


        <subject id="AI" name="Artificial Intelligence" role="WORKER">
            <description>This is a description for the AI subject. You can read it if you like. I don't care.</description>
            <malcolm:themes>
```

```
        <theme name='HCI' value='25.0' />

        <theme name='AH' value='50.0' />

        <theme name='AI' value='100.0' />

      </malcolm:themes>


      <security>

        <action name="browse" role="USER" />

        <action name="read_article" role="ADMIN" />

        <action name="submit" role="ADMIN" />

        <action name="edit_themes" role="ADMIN" />

        <action name="edit_requirements" role="ADMIN" />

        <action name="edit_links" role="ADMIN" />

        <action name="remove_article" role="ADMIN" />

      </security>


    </subject>


  </subjects>


  <metadata>


    <!--

      Metadata describing an Artilce in the site

    -->


    <namespaces>


      <!--

        List of XML namespaces used by the Metadata scheme

        e.g.

          <namespace prefix="$prefix" uri="$uri_to_scheme" />

      -->


      <namespace prefix="dc" uri="http://purl.org/DC#" />

      <namespace prefix="lom" uri="http://purl.org/LOM#" />
    <namespace prefix="malcolm" uri="http://malcolm.xtracker.co.za/#" />
</namespaces>

<fields>

    <!--
        List of fields describing an Article
        e.g.

            <field>
                <name>$name_of_field</name>
                <description>$description_of_field</description>
                <type>TEXTBOX|LIST|TEXT</type>
                <label>$label_for_field</label>
                <sqlFieldName>$sql_field_name</sqlFieldName>
                <sqlCreateStatement>$sql_create_statement</sqlCreateStatement>
                <namespace>$namespace_prefix</namespace>
                <required>true|false</required>
                <orderBy>true|false</orderBy>
                <primary>true|false</primary>
                <values default="$default_value">
                    <value>$textual_value</value>
                </values>
                <validator>
                    <type>REQUIRED|REGEX|COMPARE</type>
                    <fields>
                        <field>$html_field_name</field>
                    </fields>
                    <expression match="yes|no">$regex</expression>
                    <message>$error_message</message>
                </validator>
            </field>
    -->
```

```
<field>
   <name>title</name>
   <description>The title of the document.</description>
   <type>TEXTBOX</type>
   <label>Title</label>
   <sqlFieldName>dc_title</sqlFieldName>
   <sqlCreateStatement>dc_title varchar(100)</sqlCreateStatement>
   <namespace>dc</namespace>
   <required>true</required>
   <orderBy>true</orderBy>
   <primary>true</primary>
   <values default="" />
   <validator>
      <type>REQUIRED</type>
      <fields>
         <field>dc:title</field>
      </fields>
      <expression />
      <message>Please enter a title.</message>
   </validator>
</field>

<field>
   <name>creator</name>
   <description>The creator of the document.</description>
   <type>TEXTBOX</type>
   <label>Creator</label>
   <sqlFieldName>dc_creator</sqlFieldName>
   <sqlCreateStatement>dc_creator varchar(100)</sqlCreateStatement>
   <namespace>dc</namespace>
   <required>true</required>
   <orderBy>true</orderBy>
   <primary>true</primary>
   <values default="" />
   <validator>
```

```
        <type>REQUIRED</type>
        <fields>
            <field>dc:creator</field>
        </fields>
        <expression />
        <message>Please enter a creator.</message>
    </validator>
</field>

<field>
    <name>subject</name>
    <description>The subject of the document.</description>
    <type>LIST</type>
    <label>Subject</label>
    <sqlFieldName>dc_subject</sqlFieldName>
    <sqlCreateStatement>dc_subject varchar(100)</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>true</required>
    <orderBy>false</orderBy>
    <primary>true</primary>
    <values default="">
        <value>-----</value>
        <value>Adaptive hypermedia [AH]</value>
        <value>Human Computer Interaction [HCI]</value>
        <value>Artificial Intelligence [AI]</value>
    </values>
    <validator>
        <type>REGEX</type>
        <fields>
            <field>dc:subject</field>
        </fields>
        <expression match="no">-----</expression>
        <message>Please select a valid subject.</message>
    </validator>
</field>
```

```
<field>
   <name>description</name>
   <description>The description of the document.</description>
   <type>TEXT</type>
   <label>Description</label>
   <sqlFieldName>dc_description</sqlFieldName>
   <sqlCreateStatement>dc_description blob</sqlCreateStatement>
   <namespace>dc</namespace>
   <required>true</required>
   <orderBy>false</orderBy>
   <primary>true</primary>
   <values default="" />
   <validator>
      <type>REQUIRED</type>
      <fields>
         <field>dc:description</field>
      </fields>
      <expression />
      <message>Please enter a description for the document.</message>
   </validator>
</field>

<field>
   <name>publisher</name>
   <description>The publisher of the document.</description>
   <type>TEXTBOX</type>
   <label>Publisher</label>
   <sqlFieldName>dc_publisher</sqlFieldName>
   <sqlCreateStatement>dc_publisher varchar(100)</sqlCreateStatement>
   <namespace>dc</namespace>
   <required>false</required>
   <orderBy>true</orderBy>
   <primary>true</primary>
   <values default="" />
```

```
    <validator />
</field>

<field>
    <name>contributor</name>
    <description>The contributor of the document.</description>
    <type>TEXTBOX</type>
    <label>Contributor</label>
    <sqlFieldName>dc_contributor</sqlFieldName>
    <sqlCreateStatement>dc_contributor varchar(100)</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>false</required>
    <orderBy>false</orderBy>
    <primary>true</primary>
    <values default="" />
    <validator />
</field>

<field>
    <name>date</name>
    <description>The date of the document.</description>
    <type>TEXTBOX</type>
    <label>Date</label>
    <sqlFieldName>dc_date</sqlFieldName>
    <sqlCreateStatement>dc_date DateTime</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>true</required>
    <orderBy>true</orderBy>
    <primary>true</primary>
    <values default="" />
    <validator>
        <type>REGEX</type>
        <fields>
            <field>dc:date</field>
        </fields>
        <expression match="yes">\d\d\d\d-\d\d-\d\d</expression>
        <message>Please enter a date for the document, e.g. 2003-05-02.</message>
    </validator>
</field>

<field>
    <name>type</name>
    <description>The type of document.</description>
    <type>LIST</type>
    <label>Type</label>
    <sqlFieldName>dc_type</sqlFieldName>
    <sqlCreateStatement>dc_type varchar(32)</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>true</required>
    <orderBy>true</orderBy>
    <primary>true</primary>
    <values default="">
        <value>-----</value>
        <value>Site article</value>
    </values>
    <validator>
        <type>REGEX</type>
        <fields>
            <field>dc:type</field>
        </fields>
        <expression match="no">-----</expression>
        <message>Please select a valid type.</message>
    </validator>
</field>

<field>
    <name>format</name>
    <description>The format of document's content.</description>
    <type>LIST</type>
    <label>Format</label>
    <sqlFieldName>dc_format</sqlFieldName>
    <sqlCreateStatement>dc_format varchar(16)</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>true</required>
    <orderBy>false</orderBy>
    <primary>true</primary>
    <values default="">
        <value>-----</value>
        <value>text/xml</value>
    </values>
    <validator>
        <type>REGEX</type>
        <fields>
            <field>dc:format</field>
        </fields>
        <expression match="no">-----</expression>
        <message>Please select a valid format.</message>
    </validator>
</field>

<field>
    <name>identifier</name>
    <description>The identifier of the document.</description>
    <type>TEXTBOX</type>
    <label>Identifier</label>
    <sqlFieldName>dc_identifier</sqlFieldName>
    <sqlCreateStatement>dc_identifier varchar(16)</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>false</required>
    <orderBy>false</orderBy>
    <primary>true</primary>
    <values default="" />
    <validator />
</field>
```

```
<field>

   <name>source</name>

   <description>The original source of the document.</description>

   <type>TEXTBOX</type>

   <label>Source</label>

   <sqlFieldName>dc_source</sqlFieldName>

   <sqlCreateStatement>dc_source blob</sqlCreateStatement>

   <namespace>dc</namespace>

   <required>false</required>

   <orderBy>false</orderBy>

   <primary>true</primary>

   <values default="" />

   <validator />

</field>


<field>

   <name>language</name>

   <description>The language of document's content.</description>

   <type>LIST</type>

   <label>Language</label>

   <sqlFieldName>dc_language</sqlFieldName>

   <sqlCreateStatement>dc_language varchar(5)</sqlCreateStatement>

   <namespace>dc</namespace>

   <required>true</required>

   <orderBy>false</orderBy>

   <primary>true</primary>

   <values default="">

      <value>-----</value>

      <value>en-GB</value>

      <value>en-US</value>

   </values>

   <validator>

      <type>REGEX</type>

      <fields>

         <field>dc:language</field>
```

```
        </fields>

        <expression match="no">-----</expression>

        <message>Please select a valid language.</message>

    </validator>

</field>


<field>

    <name>coverage</name>

    <description>The coverage of the document.</description>

    <type>TEXTBOX</type>

    <label>Coverage</label>

    <sqlFieldName>dc_coverage</sqlFieldName>

    <sqlCreateStatement>dc_coverage blob</sqlCreateStatement>

    <namespace>dc</namespace>

    <required>false</required>

    <orderBy>false</orderBy>

    <primary>true</primary>

    <values default="" />

    <validator />

</field>


<field>

    <name>relation</name>

    <description>The relation of the document.</description>

    <type>TEXTBOX</type>

    <label>Relation</label>

    <sqlFieldName>dc_relation</sqlFieldName>

    <sqlCreateStatement>dc_relation blob</sqlCreateStatement>

    <namespace>dc</namespace>

    <required>false</required>

    <orderBy>false</orderBy>

    <primary>true</primary>

    <values default="" />

    <validator />

</field>
```

```
<field>
    <name>rights</name>
    <description>The rights of the document.</description>
    <type>TEXTBOX</type>
    <label>Rights</label>
    <sqlFieldName>dc_rights</sqlFieldName>
    <sqlCreateStatement>dc_rights blob</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>false</required>
    <orderBy>false</orderBy>
    <primary>true</primary>
    <values default="" />
    <validator />
</field>

<field>
    <name>requirement</name>
    <description>The requirements of the article.</description>
    <type>TEXT</type>
    <label>Requirements</label>
    <sqlFieldName>lom_requirement</sqlFieldName>
    <sqlCreateStatement>lom_requirement varchar(255)</sqlCreateStatement>
    <namespace>lom</namespace>
    <required>false</required>
    <orderBy>false</orderBy>
    <primary>true</primary>
    <values default="" />
    <validator />
</field>

<field>
    <name>difficulty</name>
    <description>The difficulty of article's content.</description>
    <type>LIST</type>

    <label>Difficulty</label>

    <sqlFieldName>lom_difficulty</sqlFieldName>

    <sqlCreateStatement>lom_difficulty varchar(10)</sqlCreateStatement>

    <namespace>lom</namespace>

    <required>true</required>

    <orderBy>false</orderBy>

    <primary>true</primary>

    <values default="">

        <value>-----</value>

        <value>Easy</value>

        <value>Medium</value>

        <value>Hard</value>

    </values>

    <validator>

        <type>REGEX</type>

        <fields>

            <field>lom:difficulty</field>

        </fields>

        <expression match="no">-----</expression>

        <message>Please select a valid difficulty level.</message>

    </validator>

</field>


<field>

    <name>keyword</name>

    <description>The keywords of the article.</description>

    <type>TEXT</type>

    <label>Keywords</label>

    <sqlFieldName>lom_keyword</sqlFieldName>

    <sqlCreateStatement>lom_keyword blob</sqlCreateStatement>

    <namespace>lom</namespace>

    <required>true</required>

    <orderBy>false</orderBy>

    <primary>true</primary>
    <values default="" />
    <validator>
        <type>REQUIRED</type>
        <fields>
            <field>lom:keyword</field>
        </fields>
        <expression />
        <message>Please enter keywords for the article.</message>
    </validator>

</field>

<field>
    <name>stretch</name>
    <description>Should this section stretch.</description>
    <type>LIST</type>
    <label>Stretch</label>
    <sqlFieldName>malcolm_stretch</sqlFieldName>
    <sqlCreateStatement>malcolm_stretch varchar(3)</sqlCreateStatement>
    <namespace>malcolm</namespace>
    <required>true</required>
    <orderBy>false</orderBy>
    <primary>false</primary>
    <values default="">
        <value>-----</value>
        <value>Yes</value>
        <value>No</value>
    </values>
    <validator>
        <type>REGEX</type>
        <fields>
            <field>malcolm:stretch</field>
        </fields>
        <expression match="no">-----</expression>
        <message>Please indicate whether this section can stretch or not.</message>
```

```
            </validator>

        </field>


        <field>

            <name>clusters</name>

            <description>The clusters to which this Article belongs.</description>

            <type>TEXTBOX</type>

            <label>Clusters</label>

            <sqlFieldName>malcolm_clusters</sqlFieldName>

            <sqlCreateStatement>malcolm_clusters varchar(255) default ''</sqlCreateStatement>

            <namespace>malcolm</namespace>

            <required>false</required>

            <orderBy>false</orderBy>

            <primary>true</primary>

            <values default="" />

            <validator />

        </field>

    </fields>

</metadata>
```

Page XML

```
<databases>

   <!--

        List of Databases and their required user levels

        e.g.

            <database id="$id" name="$name" role="DEFAULT|USER|WORKER|ADMIN|SUPER">

                <description>$description_of_database</description>

                <malcolm:themes>

                    <theme name='$name' value='$float_value' />

                </malcolm:themes>

               <security>

                    <action name="$action" role="DEFAULT|USER|WORKER|ADMIN|SUPER" />

                </security>

           </database>

    -->
    <database id="Hype" name="Hype" role="USER">
        <description>A bunch of Hype statements.</description>
         <malcolm:themes>
             <theme name='AH' value='100.0' />
             <theme name='AI' value='75.0' />
             <theme name='Datamining' value='60.0' />
         </malcolm:themes>
        <security>
             <action name="browse" role="USER" />
             <action name="view_statement" role="USER" />
             <action name="create" role="WORKER" />
             <action name="remove" role="ADMIN" />
             <action name="accept" role="ADMIN" />
         </security>
    </database>
    <database id="os" name="Open source" role="WORKER">
        <description>Statements that were made about Open Source projects and ideas.</description>
         <malcolm:themes>
             <theme name='AH' value='100.0' />
             <theme name='AI' value='75.0' />
             <theme name='Datamining' value='60.0' />
         </malcolm:themes>
        <security>
             <action name="browse" role="USER" />
             <action name="view_statement" role="USER" />
             <action name="create" role="WORKER" />
             <action name="remove" role="ADMIN" />
             <action name="accept" role="ADMIN" />
         </security>
    </database>
</databases>
<database>
    <!--
         SQL describing the various tables in the Statements database.
         e.g.
```

```
          <sql>$sql_create_statement</sql>
    -->
    <sql>CREATE TABLE ${instance_name}_Statements (
                                        ${metadata_sql},
                                     );
     </sql>
</database>
<metadata>
    <!--
         Metadata describing an Artilce in the site
     -->
    <namespaces>
        <!--
             List of XML namespaces used by the Metadata scheme
             e.g.
                 <namespace prefix="$prefix" uri="$uri_to_scheme" />
         -->
        <namespace prefix="dc" uri="http://purl.org/DC#" />
         <namespace prefix="lom" uri="http://purl.org/LOM#" />
         <namespace prefix="malcolm" uri="http://malcolm.xtracker.co.za/#" />
     </namespaces>
    <fields>
        <!--
             List of fields describing a Document
             e.g.
                <field>
                     <name>$name_of_field</name>
                     <description>$description_of_field</description>
                     <type>TEXTBOX|LIST|TEXT</type>
                     <label>$label_for_field</label>
                     <sqlFieldName>$sql_field_name</sqlFieldName>
                     <sqlCreateStatement>$sql_create_statement</sqlCreateStatement>
                     <namespace>$namespace_prefix</namespace>
                     <required>true|false</required>
                     <orderBy>true|false</orderBy>
```

```
            <primary>true|false</primary>
            <values default="$default_value">
               <value>$textual_value</value>
            </values>
            <validator>
               <type>REQUIRED|REGEX|COMPARE</type>
               <fields>
                  <field>$html_field_name</field>
               </fields>
               <expression match="yes|no">$regex</expression>
               <message>$error_message</message>
            </validator>
         </field>
 -->
<field>
    <name>statement</name>
    <description>The actual hype statement.</description>
    <type>TEXT</type>
    <label>Statement</label>
    <sqlFieldName>malcolm_statement</sqlFieldName>
    <sqlCreateStatement>malcolm_statement blob</sqlCreateStatement>
    <namespace>malcolm</namespace>
    <required>true</required>
    <orderBy>true</orderBy>
    <primary>true</primary>
    <values default="" />
    <validator>
       <type>REQUIRED</type>
       <fields>
          <field>malcolm:statement</field>
       </fields>
       <expression />
       <message>Please enter the hype statement.</message>
    </validator>
 </field>
```

```
<field>
    <name>creator</name>
    <description>The person who made the statement.</description>
    <type>TEXTBOX</type>
    <label>Creator</label>
    <sqlFieldName>dc_creator</sqlFieldName>
    <sqlCreateStatement>dc_creator varchar(100)</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>true</required>
    <orderBy>true</orderBy>
    <primary>true</primary>
    <values default="" />
    <validator>
        <type>REQUIRED</type>
        <fields>
            <field>dc:creator</field>
        </fields>
        <expression />
        <message>Please enter the person who made the statement.</message>
    </validator>
 </field>
<field>
    <name>date</name>
    <description>The date of the document.</description>
    <type>TEXTBOX</type>
    <label>Date</label>
    <sqlFieldName>dc_date</sqlFieldName>
    <sqlCreateStatement>dc_date Date default now()</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>true</required>
    <orderBy>true</orderBy>
    <primary>true</primary>
    <values default="" />
    <validator>
        <type>REGEX</type>
        <fields>
            <field>dc:date</field>
        </fields>
        <expression match="yes">\d\d\d\d-\d\d-\d\d</expression>
        <message>Please enter a date for the document, e.g. 2003-05-02.</message>
    </validator>
 </field>
<field>
    <name>identifier</name>
    <description>The identifier of the document.</description>
    <type>TEXTBOX</type>
    <label>Identifier</label>
    <sqlFieldName>dc_identifier</sqlFieldName>
    <sqlCreateStatement>dc_identifier varchar(36)</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>false</required>
    <orderBy>false</orderBy>
    <primary>false</primary>
    <values default="" />
    <validator />
 </field>
<field>
    <name>source</name>
    <description>The original source of the document.</description>
    <type>TEXT</type>
    <label>Source</label>
    <sqlFieldName>dc_source</sqlFieldName>
    <sqlCreateStatement>dc_source blob</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>true</required>
    <orderBy>false</orderBy>
    <primary>true</primary>
    <values default="" />
    <validator>
        <type>REQUIRED</type>
        <fields>
            <field>dc:source</field>
        </fields>
        <expression />
        <message>Please enter the original source.</message>
    </validator>
</field>
<field>
    <name>rights</name>
    <description>The rights of the document.</description>
    <type>TEXTBOX</type>
    <label>Rights</label>
    <sqlFieldName>dc_rights</sqlFieldName>
    <sqlCreateStatement>dc_rights blob</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>false</required>
    <orderBy>false</orderBy>
    <primary>true</primary>
    <values default="" />
    <validator />
 </field>
<field>
    <name>submittedBy</name>
    <description>The user who submitted the statement.</description>
    <type>TEXTBOX</type>
    <label>Submitted by</label>
    <sqlFieldName>malcolm_submittedBy</sqlFieldName>
    <sqlCreateStatement>malcolm_submittedBy varchar(16)</sqlCreateStatement>
    <namespace>malcolm</namespace>
    <required>true</required>
    <orderBy>true</orderBy>
    <primary>true</primary>
    <values default="" />
    <validator>
        <type>REQUIRED</type>
```

```
            <fields>
               <field>malcolm:submittedBy</field>
            </fields>
            <expression />
            <message>Please enter the your name.</message>
         </validator>
   </field>
   <field>
         <name>accepted</name>
         <description>Indicates whether the statement has been accepted.</description>
         <type>TEXTBOX</type>
         <label>Accepted</label>
         <sqlFieldName>malcolm_accepted</sqlFieldName>
         <sqlCreateStatement>malcolm_accepted tinyint(1) default 0</sqlCreateStatement>
         <namespace>malcolm</namespace>
         <required>false</required>
         <orderBy>false</orderBy>
         <primary>false</primary>
         <values default="" />
         <validator />
    </field>
   <field>
         <name>notes</name>
         <description>Some additional notes on the statement.</description>
         <type>TEXT</type>
         <label>Notes</label>
         <sqlFieldName>malcolm_notes</sqlFieldName>
         <sqlCreateStatement>malcolm_notes blob</sqlCreateStatement>
         <namespace>malcolm</namespace>
         <required>false</required>
         <orderBy>false</orderBy>
         <primary>true</primary>
         <values default="" />
         <validator />
    </field>
   <field>
         <name>database</name>
         <description>The database containing the statement.</description>
         <type>TEXT</type>
         <label>Database</label>
         <sqlFieldName>malcolm_database</sqlFieldName>
         <sqlCreateStatement>malcolm_database varchar(50)</sqlCreateStatement>
         <namespace>malcolm</namespace>
         <required>false</required>
         <orderBy>false</orderBy>
         <primary>false</primary>
         <values default="" />
         <validator />
    </field>
  </fields>
</metadata>
```

Discussion Forum configuration XML

```
<topics>
  <!--
      Topics available in the online discussion forum
      e.g.
        <topic id="$id" name="$name_of_topic" role="DEFAULT|USER|WORKER|ADMIN|SUPER">
            <description>$description_of_topic</description>
            <malcolm:themes>
                <theme name='$name' value='$float_value' />
            </malcolm:themes>
           e.g
            <action name="$action" role="DEFAULT|USER|WORKER|ADMIN|SUPER" />
        </topic>
  -->
  <topic id="General" name="General discussions" role="USER">
      <description>Here you can post messages that do not relate to any specific area of research.</description>
      <malcolm:themes />
     <security>
```

```
        <action name="view_message" role="USER" />
         <action name="reply" role="USER" />
         <action name="post" role="USER" />
         <action name="view_messages" role="USER" />
         <action name="view_replies" role="USER" />
         <action name="search" role="USER" />
      </security>
    </topic>
    <topic id="AH" name="Adaptive Hypermedia" role="WORKER">
       <description>This topic is for messages relating to Adaptive Hypermedia. For instance methods of
adaptation of content and presentation.</description>
       <malcolm:themes>
         <theme name='AH' value='100.0' />
         <theme name='AI' value='75.0' />
         <theme name='Datamining' value='60.0' />
       </malcolm:themes>
      <security>
        <action name="view_message" role="WORKER" />
         <action name="reply" role="WORKER" />
         <action name="post" role="WORKER" />
         <action name="view_messages" role="WORKER" />
         <action name="view_replies" role="WORKER" />
         <action name="search" role="WORKER" />
      </security>
    </topic>
    <topic id="AI" name="Artificial Intelligence" role="USER">
       <description>Discussion board for everything relating to Artificial Intelligence.</description>
       <malcolm:themes>
         <theme name='AI' value='100.0' />
       </malcolm:themes>
      <security>
        <action name="view_message" role="USER" />
         <action name="reply" role="USER" />
         <action name="post" role="USER" />
         <action name="view_messages" role="USER" />
```

```
            <action name="view_replies" role="USER" />

            <action name="search" role="USER" />

          </security>

        </topic>

        <topic id="Hype" name="Hype" role="USER">

           <description>Discussion board for everything relating to Hype statements.</description>

           <malcolm:themes />

          <security>

            <action name="view_message" role="USER" />

            <action name="reply" role="USER" />

            <action name="post" role="USER" />

            <action name="view_messages" role="USER" />

            <action name="view_replies" role="USER" />

            <action name="search" role="USER" />

          </security>

        </topic>

      </topics>
```

Repositories configuration XML

```
      <repository id="Library" title="Digital Library" role="DEFAULT" renderMetadata="true" renderContent="true"
visible="true">

          <description><![CDATA[

           This is the IKS Resource database. <a
href="/jsp/modules/repository/index.jsp?repository={1}&amp;action={2}&amp;id={4}">Enter the database</a>.

          ]]>

           </description>

         <security default="DEFAULT">

            <action name="edit" role="ADMIN" />

            <action name="submit" role="ADMIN" />

            <action name="revise" role="ADMIN" />

            <action name="remove" role="ADMIN" />

            <action name="accept" role="ADMIN" />

            <action name="unlock" role="ADMIN" />

            <action name="create_directory" role="ADMIN" />

            <action name="remove_directory" role="ADMIN" />

            <action name="view_files" role="DEFAULT" />
```

```
    <action name="view_folders" role="DEFAULT" />

    <action name="view_file" role="DEFAULT" />

    <action name="search" role="DEFAULT" />

</security>

<metadata>

   <!--

        Metadata describing a Document in the site

    -->

   <namespaces>

      <!--

          List of XML namespaces used by the Metadata scheme

          e.g.

              <namespace prefix="$prefix" uri="$uri_to_scheme" />

       -->

      <namespace prefix="dc" uri="http://purl.org/DC#" />

       <namespace prefix="lom" uri="http://purl.org/LOM#" />

       <namespace prefix="malcolm" uri="http://malcolm.xtracker.co.za/#" />

    </namespaces>

   <fields>

      <!--

          List of fields describing a Document

          e.g.

             <field>

                 <name>$name_of_field</name>

                 <description>$description_of_field</description>

                 <type>TEXTBOX|LIST|TEXT</type>

                 <label>$label_for_field</label>

                 <sqlFieldName>$sql_field_name</sqlFieldName>

                 <sqlCreateStatement>$sql_create_statement</sqlCreateStatement>

                 <namespace>$namespace_prefix</namespace>

                 <required>true|false</required>

                 <orderBy>true|false</orderBy>

                 <primary>true|false</primary>

                 <values default="$default_value">

                    <value>$textual_value</value>
                    </values>
                    <validator>
                        <type>REQUIRED|REGEX|COMPARE</type>
                        <fields>
                            <field>$html_field_name</field>
                        </fields>
                        <expression match="yes|no">$regex</expression>
                        <message>$error_message</message>
                        <script><![CDATA[function $name_click(field, value)
{ window.open("/jsp/modules/repository/scripts/projects.jsp?repository=$repository_name&field=" +field+ "&value="
+value, "_new", "width=750, height=500, status=no, toolbar=no"); }]]></script>
                    </validator>
                </field>
            -->
            <field>
                <name>parent</name>
                <description>The parent folder of the document.</description>
                <type>TEXTBOX</type>
                <label>Parent folder</label>
                <sqlFieldName>malcolm_parent</sqlFieldName>
                <sqlCreateStatement>malcolm_parent varchar(36)</sqlCreateStatement>
                <namespace>malcolm</namespace>
                <required>false</required>
                <orderBy>false</orderBy>
                <primary>false</primary>
                <values default="" />
                <validator />
            </field>
            <field>
                <name>path</name>
                <description>The path on the server to the actual document.</description>
                <type>TEXTBOX</type>
                <label>Path</label>
                <sqlFieldName>malcolm_path</sqlFieldName>
                <sqlCreateStatement>malcolm_path varchar(255)</sqlCreateStatement>
                <namespace>malcolm</namespace>
```

```
    <required>false</required>

    <orderBy>false</orderBy>

    <primary>false</primary>

    <values default="" />

    <validator />

 </field>

<field>

    <name>submittedBy</name>

    <description>The username of the user who submitted the document.</description>

    <type>TEXTBOX</type>

    <label>Submitted by</label>

    <sqlFieldName>malcolm_submittedBy</sqlFieldName>

    <sqlCreateStatement>malcolm_submittedBy varchar(15)</sqlCreateStatement>

    <namespace>malcolm</namespace>

    <required>true</required>

    <orderBy>true</orderBy>

    <primary>false</primary>

    <values default="" />

    <validator />

 </field>

<field>

    <name>modifiedBy</name>

    <description>The username of the person who modified the document.</description>

    <type>TEXTBOX</type>

    <label>Modified by</label>

    <sqlFieldName>malcolm_modifiedBy</sqlFieldName>

    <sqlCreateStatement>malcolm_modifiedBy varchar(36)</sqlCreateStatement>

    <namespace>malcolm</namespace>

    <required>false</required>

    <orderBy>false</orderBy>

    <primary>false</primary>

    <values default="" />

    <validator />

 </field>

<field>
```

```
    <name>locked</name>
    <description>Indicates whether the document is locked for editing.</description>
    <type>TEXTBOX</type>
    <label>Locked</label>
    <sqlFieldName>malcolm_locked</sqlFieldName>
    <sqlCreateStatement>malcolm_locked tinyint(1)</sqlCreateStatement>
    <namespace>malcolm</namespace>
    <required>false</required>
    <orderBy>false</orderBy>
    <primary>false</primary>
    <values default="" />
    <validator />
 </field>
<field>
    <name>notifyList</name>
    <description>The list of Users to be notified when the document becomes unlocked.</description>
    <type>TEXTBOX</type>
    <label>Notify list</label>
    <sqlFieldName>malcolm_notifyList</sqlFieldName>
    <sqlCreateStatement>malcolm_notifyList varchar(255)</sqlCreateStatement>
    <namespace>malcolm</namespace>
    <required>false</required>
    <orderBy>false</orderBy>
    <primary>false</primary>
    <values default="" />
    <validator />
 </field>
<field>
    <name>accepted</name>
    <description>Indicates whether the document has been accepted by a moderator.</description>
    <type>TEXTBOX</type>
    <label>Accepted</label>
    <sqlFieldName>malcolm_accepted</sqlFieldName>
    <sqlCreateStatement>malcolm_accepted tinyint(1)</sqlCreateStatement>
    <namespace>malcolm</namespace>
    <required>false</required>
    <orderBy>false</orderBy>
    <primary>false</primary>
    <values default="" />
    <validator />
 </field>
<field>
    <name>version</name>
    <description>The revision number of this document.</description>
    <type>TEXTBOX</type>
    <label>Version</label>
    <sqlFieldName>malcolm_version</sqlFieldName>
    <sqlCreateStatement>malcolm_version varchar(16) default 'Original'</sqlCreateStatement>
    <namespace>malcolm</namespace>
    <required>true</required>
    <orderBy>false</orderBy>
    <primary>true</primary>
    <values default="Original" />
    <validator>
        <type>REGEX</type>
        <fields>
            <field>malcolm:version</field>
        </fields>
        <expression match="yes">^(?:Original|(?:\d+\.?)+)$</expression>
        <message>The version must either be Original or some numeric value, e.g. 1 or 1.1</message>
    </validator>
 </field>
<field>
    <name>status</name>
    <description>The current status of the document.</description>
    <type>LIST</type>
    <label>Status</label>
    <sqlFieldName>malcolm_status</sqlFieldName>
    <sqlCreateStatement>malcolm_status varchar(16)</sqlCreateStatement>
    <namespace>malcolm</namespace>
```

```
          <required>true</required>

          <orderBy>false</orderBy>

          <primary>true</primary>

          <values default="">

              <value>-----</value>

              <value>Draft</value>

              <value>Under review</value>

              <value>Published</value>

          </values>

          <validator>

              <type>REGEX</type>

              <fields>

                  <field>malcolm:status</field>

              </fields>

              <expression match="no">-----</expression>

              <message>Please select the current status of the document.</message>

          </validator>

      </field>

     <field>

          <name>visibility</name>

          <description>Should the document be visible outside of this research group.</description>

          <type>LIST</type>

          <label>Visibility</label>

          <sqlFieldName>malcolm_visibility</sqlFieldName>

          <sqlCreateStatement>malcolm_visibility enum('Private', 'Public') default
'Private'</sqlCreateStatement>

          <namespace>malcolm</namespace>

          <required>true</required>

          <orderBy>false</orderBy>

          <primary>true</primary>

          <values default="Public">

              <value>-----</value>

              <value>Private</value>

              <value>Public</value>

          </values>
```

```
        <validator>
            <type>REGEX</type>
            <fields>
                <field>malcolm:visibility</field>
            </fields>
            <expression match="no">-----</expression>
            <message>Please select visibility of the document.</message>
        </validator>
     </field>
    <field>
        <name>instance</name>
        <description>The research project to which this document belongs.</description>
        <type>TEXTBOX</type>
        <label>Parent project</label>
        <sqlFieldName>malcolm_instance</sqlFieldName>
        <sqlCreateStatement>malcolm_instance varchar(16)</sqlCreateStatement>
        <namespace>malcolm</namespace>
        <required>false</required>
        <orderBy>false</orderBy>
        <primary>false</primary>
        <values default="" />
        <validator />
     </field>
    <field>
        <name>role</name>
        <description>The required User role to access this document.</description>
        <type>TEXTBOX</type>
        <label>Required role</label>
        <sqlFieldName>malcolm_role</sqlFieldName>
        <sqlCreateStatement>malcolm_role int</sqlCreateStatement>
        <namespace>malcolm</namespace>
        <required>false</required>
        <orderBy>false</orderBy>
        <primary>false</primary>
        <values default="" />
        <validator />
     </field>
    <field>
        <name>dateModified</name>
        <description>The date on which the document was modified.</description>
        <type>TEXTBOX</type>
        <label>Date modified</label>
        <sqlFieldName>malcolm_dateModified</sqlFieldName>
        <sqlCreateStatement>malcolm_dateModified varchar(10)</sqlCreateStatement>
        <namespace>malcolm</namespace>
        <required>false</required>
        <orderBy>true</orderBy>
        <primary>false</primary>
        <values default="" />
        <validator />
     </field>
    <field>
        <name>title</name>
        <description>The name given to the resource by the CREATOR or PUBLISHER.</description>
        <type>TEXTBOX</type>
        <label>Title</label>
        <sqlFieldName>dc_title</sqlFieldName>
        <sqlCreateStatement>dc_title varchar(255)</sqlCreateStatement>
        <namespace>dc</namespace>
        <required>true</required>
        <orderBy>true</orderBy>
        <primary>true</primary>
        <values default="" />
        <validator>
            <type>REQUIRED</type>
            <fields>
                <field>dc:title</field>
            </fields>
            <expression />
            <message>Please enter a title.</message>
```

```xml
        </validator>
    </field>
    <field>
        <name>creator</name>
        <description>The person or organization primarily responsible for creating the intellectual content of the resource. For example, authors in the case of written documents, artists, photographers, or illustrators in the case of visual resources.</description>
        <type>TEXTBOX</type>
        <label>Creator</label>
        <sqlFieldName>dc_creator</sqlFieldName>
        <sqlCreateStatement>dc_creator varchar(100)</sqlCreateStatement>
        <namespace>dc</namespace>
        <required>true</required>
        <orderBy>true</orderBy>
        <primary>true</primary>
        <values default="" />
        <validator>
            <type>REGEX</type>
            <fields>
                <field>dc:creator</field>
            </fields>
            <expression match="yes">^\w+,\s*(?:\w+(?:\s\w+)*)*(\s(\w\.\s*)+)*$</expression>
            <message>Please enter a creator in the format Surname, Name or Surname, A.B.C.</message>
        </validator>
    </field>
    <field>
        <name>subject</name>
        <description>The topic of the resource. Typically, subject will be expressed as keywords or phrases that describe the subject or content of the resource. The use of controlled vocabularies and formal classification schemas is encouraged.</description>
        <type>LIST</type>
        <label>Subject</label>
        <sqlFieldName>dc_subject</sqlFieldName>
        <sqlCreateStatement>dc_subject varchar(100)</sqlCreateStatement>
        <namespace>dc</namespace>
        <required>true</required>
        <orderBy>false</orderBy>

        <primary>true</primary>

        <values default="">

            <value>-----</value>

            <value>E-business</value>

            <value>E-learning</value>

            <value>Portals</value>

            <value>Research issues</value>

            <value>Technical developments</value>

        </values>

        <validator>

            <type>REGEX</type>

            <fields>

                <field>dc:subject</field>

            </fields>

            <expression match="no">-----</expression>

            <message>Please select a valid subject.</message>

        </validator>

    </field>

    <field>

        <name>description</name>

        <description>A textual description of the content of the resource, including abstracts in the case of
document-like objects or content descriptions in the case of visual resources.</description>

        <type>TEXT</type>

        <label>Description</label>

        <sqlFieldName>dc_description</sqlFieldName>

        <sqlCreateStatement>dc_description blob</sqlCreateStatement>

        <namespace>dc</namespace>

        <required>true</required>

        <orderBy>false</orderBy>

        <primary>true</primary>

        <values default="" />

        <validator>

            <type>REQUIRED</type>

            <fields>
```

```
                <field>dc:description</field>
            </fields>
            <expression />
            <message>Please enter a description for the document.</message>
        </validator>
    </field>
    <field>
        <name>keywords</name>
        <description>Keywords describing the document.</description>
        <type>TEXTBOX</type>
        <label>Keywords</label>
        <sqlFieldName>malcolm_keywords</sqlFieldName>
        <sqlCreateStatement>malcolm_keywords varchar(255)</sqlCreateStatement>
        <namespace>malcolm</namespace>
        <required>false</required>
        <orderBy>false</orderBy>
        <primary>true</primary>
        <values default="" />
        <validator />
    </field>
    <field>
        <name>publisher</name>
        <description>The entity responsible for making the resource available in its present form, such as a
publishing house, a university department, or a corporate entity.</description>
        <type>TEXTBOX</type>
        <label>Publisher</label>
        <sqlFieldName>dc_publisher</sqlFieldName>
        <sqlCreateStatement>dc_publisher varchar(100)</sqlCreateStatement>
        <namespace>dc</namespace>
        <required>false</required>
        <orderBy>true</orderBy>
        <primary>true</primary>
        <values default="" />
        <validator />
    </field>
```

```
<field>
    <name>contributor</name>
    <description>A person or organization not specified in a CREATOR element who has made significant intellectual contributions to the resource but whose contribution is secondary to any person or organization specified in a CREATOR element (for example, editor, transcriber, and illustrator).</description>
    <type>TEXTBOX</type>
    <label>Contributor</label>
    <sqlFieldName>dc_contributor</sqlFieldName>
    <sqlCreateStatement>dc_contributor varchar(100)</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>false</required>
    <orderBy>false</orderBy>
    <primary>true</primary>
    <values default="" />
    <validator />
 </field>
<field>
    <name>date</name>
    <description>The date the resource was made available in its present form. Recommended best practice is an 8 digit number in the form YYYY-MM-DD as defined in http://www.w3.org/TR/NOTE-datetime, a profile of ISO 8601. In this scheme, the date element 1994-11-05 corresponds to November 5, 1994. Many other schema are possible, but if used, they should be identified in an unambiguous manner.</description>
    <type>TEXTBOX</type>
    <label>Date</label>
    <sqlFieldName>dc_date</sqlFieldName>
    <sqlCreateStatement>dc_date Date</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>true</required>
    <orderBy>true</orderBy>
    <primary>true</primary>
    <values default="" />
    <validator>
        <type>REGEX</type>
        <fields>
            <field>dc:date</field>
        </fields>
        <expression match="yes">\d\d\d\d-\d\d-\d\d</expression>
```

```
            <message>Please enter a date for the document, e.g. 2003-05-02.</message>
        </validator>
        ]]></script>
    </field>
    <field>
        <name>type</name>
        <description>The category of the resource, such as home page, novel, poem, working paper, technical report, essay, dictionary. For the sake of interoperability, TYPE should be selected from an enumerated list that is under development in the workshop series at the time of publication of this document. See http://sunsite.berkeley.edu/Metadata/types.html for current thinking on the application of this element.</description>
        <type>LIST</type>
        <label>Type</label>
        <sqlFieldName>dc_type</sqlFieldName>
        <sqlCreateStatement>dc_type varchar(32)</sqlCreateStatement>
        <namespace>dc</namespace>
        <required>true</required>
        <orderBy>true</orderBy>
        <primary>true</primary>
        <values default="">
            <value>-----</value>
            <value>Conference paper</value>
            <value>Something else</value>
        </values>
        <validator>
            <type>REGEX</type>
            <fields>
                <field>dc:type</field>
            </fields>
            <expression match="no">-----</expression>
            <message>Please select a valid type.</message>
        </validator>
    </field>
    <field>
        <name>format</name>
        <description>The data format of the resource, used to identify the software and possibly hardware that might be needed to display or operate the resource. For the sake of interoperability, FORMAT should be selected from an enumerated list that is under development in the workshop series at the time of publication of this
```

```
document.</description>
                              <type>LIST</type>
                              <label>Format</label>
                              <sqlFieldName>dc_format</sqlFieldName>
                              <sqlCreateStatement>dc_format varchar(36)</sqlCreateStatement>
                              <namespace>dc</namespace>
                              <required>true</required>
                              <orderBy>false</orderBy>
                              <primary>false</primary>
                              <values default="">
                                <value>-----</value>
                                <value>application/msword</value>
                                <value>application/pdf</value>
                                <value>application/postscript</value>
                                <value>application/x-gzip</value>
                                <value>application/x-tar</value>
                                <value>application/x-tcl</value>
                                <value>application/x-tex</value>
                                <value>application/xml</value>
                                <value>application/zip</value>
                                <value>audio/mpeg</value>
                                <value>audio/x-wav</value>
                                <value>image/bmp</value>
                                <value>image/gif</value>
                                <value>image/jpeg</value>
                                <value>image/png</value>
                                <value>image/tiff</value>
                                <value>text/html</value>
                                <value>text/plain</value>
                                <value>video/mpeg</value>
                              </values>
                              <validator>
                                <type>REGEX</type>
                                <fields>
                                  <field>dc:format</field>
```

```
            </fields>
            <expression match="no">-----</expression>
            <message>Please select a valid format.</message>
        </validator>
    </field>
    <field>
        <name>identifier</name>
        <description>String or number used to uniquely identify the resource. Examples for networked
resources include URLs and URNs (when implemented). Other globally-unique identifiers,such as International Standard
Book Numbers (ISBN) or other formal names would also be candidates for this element in the case of off-line
resources.</description>
        <type>TEXTBOX</type>
        <label>Identifier</label>
        <sqlFieldName>dc_identifier</sqlFieldName>
        <sqlCreateStatement>dc_identifier varchar(36) PRIMARY KEY</sqlCreateStatement>
        <namespace>dc</namespace>
        <required>false</required>
        <orderBy>false</orderBy>
        <primary>false</primary>
        <values default="" />
        <validator />
    </field>
    <field>
        <name>source</name>
        <description>A string or number used to uniquely identify the work from which this resource was
derived, if applicable. For example, a PDF version of a novel might have a SOURCE element containing an ISBN number
for the physical book from which the PDF version was derived.</description>
        <type>TEXT</type>
        <label>Source</label>
        <sqlFieldName>dc_source</sqlFieldName>
        <sqlCreateStatement>dc_source blob</sqlCreateStatement>
        <namespace>dc</namespace>
        <required>false</required>
        <orderBy>false</orderBy>
        <primary>true</primary>
        <values default="" />
        <validator />
```

```
 </field>
<field>
    <name>language</name>
    <description>Language(s) of the intellectual content of the resource. Where practical, the content of this field should coincide with RFC 1766. See: ftp://ftp.isi.edu/in-notes/rfc1766.txt.</description>
    <type>LIST</type>
    <label>Language</label>
    <sqlFieldName>dc_language</sqlFieldName>
    <sqlCreateStatement>dc_language varchar(5)</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>true</required>
    <orderBy>false</orderBy>
    <primary>true</primary>
    <values default="">
        <value>-----</value>
        <value>en-GB</value>
        <value>en-US</value>
    </values>
    <validator>
        <type>REGEX</type>
        <fields>
            <field>dc:language</field>
        </fields>
        <expression match="no">-----</expression>
        <message>Please select a valid language.</message>
    </validator>
 </field>
<field>
    <name>coverage</name>
    <description>The spatial and/or temporal characteristics of the resource. Formal specification of COVERAGE is currently under development. Users and developers should understand that use of this element is currently considered to be experimental.</description>
    <type>TEXTBOX</type>
    <label>Coverage</label>
    <sqlFieldName>dc_coverage</sqlFieldName>
    <sqlCreateStatement>dc_coverage blob</sqlCreateStatement>
```

```
            <namespace>dc</namespace>

            <required>false</required>

            <orderBy>false</orderBy>

            <primary>true</primary>

            <values default="" />

            <validator />

        </field>

        <field browse="true">

            <name>relation</name>

            <description>The relationship of this resource to other resources. The intent of this element is to
provide a means to express relationships among resources that have formal relationships to others, but exist as
discrete resources themselves. For example, images in a document, chapters in a book, or items in a collection. Formal
specification of RELATION is currently under development. Users and developers should understand that use of this
element is currently considered to be experimental.</description>

            <type>TEXTBOX</type>

            <label>Relation</label>

            <sqlFieldName>dc_relation</sqlFieldName>

            <sqlCreateStatement>dc_relation blob</sqlCreateStatement>

            <namespace>dc</namespace>

            <required>false</required>

            <orderBy>false</orderBy>

            <primary>true</primary>

            <values default="" />

            <validator />

            <script><![CDATA[function relation_click(field, value)
{ window.open("/jsp/modules/repository/scripts/relation.jsp?repository=Library&field=" +field+ "&value=" +value,
"_new", "width=750, height=500, status=no, toolbar=no"); }]]></script>

        </field>

        <field>

            <name>rights</name>

            <description>A link to a copyright notice, to a rights-management statement, or to a service that
would provide information about terms of access to the resource. Formal specification of RIGHTS is currently under
development. Users and developers should understand that use of this element is currently considered to be
experimental.</description>

            <type>TEXTBOX</type>

            <label>Rights</label>

            <sqlFieldName>dc_rights</sqlFieldName>

            <sqlCreateStatement>dc_rights blob</sqlCreateStatement>
```

```
                <namespace>dc</namespace>
                <required>false</required>
                <orderBy>false</orderBy>
                <primary>true</primary>
                <values default="" />
                <validator />
              </field>
            </fields>
          </metadata>
        </repository>
```

Statement Database configuration XML

```
<databases>
  <!--
    List of Databases and their required user levels
    e.g.
      <database id="$id" name="$name" role="DEFAULT|USER|WORKER|ADMIN|SUPER">
        <description>$description_of_database</description>
         <malcolm:themes>
           <theme name='$name' value='$float_value' />
         </malcolm:themes>
        <security>
           <action name="$action" role="DEFAULT|USER|WORKER|ADMIN|SUPER" />
         </security>
      </database>
  -->
  <database id="Hype" name="Hype" role="USER">
    <description>A bunch of Hype statements.</description>
     <malcolm:themes>
       <theme name='AH' value='100.0' />
       <theme name='AI' value='75.0' />
       <theme name='Datamining' value='60.0' />
     </malcolm:themes>
    <security>
       <action name="browse" role="USER" />
```

```
            <action name="view_statement" role="USER" />
            <action name="create" role="WORKER" />
            <action name="remove" role="ADMIN" />
            <action name="accept" role="ADMIN" />
        </security>
    </database>
    <database id="os" name="Open source" role="WORKER">
        <description>Statements that were made about Open Source projects and ideas.</description>
        <malcolm:themes>
            <theme name='AH' value='100.0' />
            <theme name='AI' value='75.0' />
            <theme name='Datamining' value='60.0' />
        </malcolm:themes>
        <security>
            <action name="browse" role="USER" />
            <action name="view_statement" role="USER" />
            <action name="create" role="WORKER" />
            <action name="remove" role="ADMIN" />
            <action name="accept" role="ADMIN" />
        </security>
    </database>
</databases>
<database>
    <!--
        SQL describing the various tables in the Statements database.
        e.g.
            <sql>$sql_create_statement</sql>
    -->
    <sql>CREATE TABLE ${instance_name}_Statements (
                                        ${metadata_sql},
                                      );
     </sql>
</database>
<metadata>
    <!--
    Metadata describing an Artilce in the site
 -->
<namespaces>
   <!--
       List of XML namespaces used by the Metadata scheme
       e.g.
           <namespace prefix="$prefix" uri="$uri_to_scheme" />
    -->
   <namespace prefix="dc" uri="http://purl.org/DC#" />
    <namespace prefix="lom" uri="http://purl.org/LOM#" />
    <namespace prefix="malcolm" uri="http://malcolm.xtracker.co.za/#" />
 </namespaces>
<fields>
   <!--
       List of fields describing a Document
       e.g.
          <field>
               <name>$name_of_field</name>
               <description>$description_of_field</description>
               <type>TEXTBOX|LIST|TEXT</type>
               <label>$label_for_field</label>
               <sqlFieldName>$sql_field_name</sqlFieldName>
               <sqlCreateStatement>$sql_create_statement</sqlCreateStatement>
               <namespace>$namespace_prefix</namespace>
               <required>true|false</required>
               <orderBy>true|false</orderBy>
               <primary>true|false</primary>
               <values default="$default_value">
                   <value>$textual_value</value>
               </values>
               <validator>
                   <type>REQUIRED|REGEX|COMPARE</type>
                   <fields>
                       <field>$html_field_name</field>
                   </fields>
```

```
                <expression match="yes|no">$regex</expression>
                <message>$error_message</message>
            </validator>
        </field>
 -->
<field>
     <name>statement</name>
     <description>The actual hype statement.</description>
     <type>TEXT</type>
     <label>Statement</label>
     <sqlFieldName>malcolm_statement</sqlFieldName>
     <sqlCreateStatement>malcolm_statement blob</sqlCreateStatement>
     <namespace>malcolm</namespace>
     <required>true</required>
     <orderBy>true</orderBy>
     <primary>true</primary>
     <values default="" />
     <validator>
         <type>REQUIRED</type>
         <fields>
             <field>malcolm:statement</field>
         </fields>
         <expression />
         <message>Please enter the hype statement.</message>
     </validator>
 </field>
<field>
     <name>creator</name>
     <description>The person who made the statement.</description>
     <type>TEXTBOX</type>
     <label>Creator</label>
     <sqlFieldName>dc_creator</sqlFieldName>
     <sqlCreateStatement>dc_creator varchar(100)</sqlCreateStatement>
     <namespace>dc</namespace>
     <required>true</required>
```

```
    <orderBy>true</orderBy>

    <primary>true</primary>

    <values default="" />

    <validator>

        <type>REQUIRED</type>

        <fields>

            <field>dc:creator</field>

        </fields>

        <expression />

        <message>Please enter the person who made the statement.</message>

    </validator>

 </field>

<field>

    <name>date</name>

    <description>The date of the document.</description>

    <type>TEXTBOX</type>

    <label>Date</label>

    <sqlFieldName>dc_date</sqlFieldName>

    <sqlCreateStatement>dc_date Date default now()</sqlCreateStatement>

    <namespace>dc</namespace>

    <required>true</required>

    <orderBy>true</orderBy>

    <primary>true</primary>

    <values default="" />

    <validator>

        <type>REGEX</type>

        <fields>

            <field>dc:date</field>

        </fields>

        <expression match="yes">\d\d\d\d-\d\d-\d\d</expression>

        <message>Please enter a date for the document, e.g. 2003-05-02.</message>

    </validator>

 </field>

<field>

    <name>identifier</name>
    <description>The identifier of the document.</description>
    <type>TEXTBOX</type>
    <label>Identifier</label>
    <sqlFieldName>dc_identifier</sqlFieldName>
    <sqlCreateStatement>dc_identifier varchar(36)</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>false</required>
    <orderBy>false</orderBy>
    <primary>false</primary>
    <values default="" />
    <validator />
 </field>
<field>
    <name>source</name>
    <description>The original source of the document.</description>
    <type>TEXT</type>
    <label>Source</label>
    <sqlFieldName>dc_source</sqlFieldName>
    <sqlCreateStatement>dc_source blob</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>true</required>
    <orderBy>false</orderBy>
    <primary>true</primary>
    <values default="" />
    <validator>
        <type>REQUIRED</type>
        <fields>
            <field>dc:source</field>
        </fields>
        <expression />
        <message>Please enter the original source.</message>
    </validator>
</field>
<field>
    <name>rights</name>
    <description>The rights of the document.</description>
    <type>TEXTBOX</type>
    <label>Rights</label>
    <sqlFieldName>dc_rights</sqlFieldName>
    <sqlCreateStatement>dc_rights blob</sqlCreateStatement>
    <namespace>dc</namespace>
    <required>false</required>
    <orderBy>false</orderBy>
    <primary>true</primary>
    <values default="" />
    <validator />
 </field>
<field>
    <name>submittedBy</name>
    <description>The user who submitted the statement.</description>
    <type>TEXTBOX</type>
    <label>Submitted by</label>
    <sqlFieldName>malcolm_submittedBy</sqlFieldName>
    <sqlCreateStatement>malcolm_submittedBy varchar(16)</sqlCreateStatement>
    <namespace>malcolm</namespace>
    <required>true</required>
    <orderBy>true</orderBy>
    <primary>true</primary>
    <values default="" />
    <validator>
        <type>REQUIRED</type>
        <fields>
            <field>malcolm:submittedBy</field>
        </fields>
        <expression />
        <message>Please enter the your name.</message>
    </validator>
</field>
<field>
    <name>accepted</name>
```

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

```
    <description>Indicates whether the statement has been accepted.</description>
    <type>TEXTBOX</type>
    <label>Accepted</label>
    <sqlFieldName>malcolm_accepted</sqlFieldName>
    <sqlCreateStatement>malcolm_accepted tinyint(1) default 0</sqlCreateStatement>
    <namespace>malcolm</namespace>
    <required>false</required>
    <orderBy>false</orderBy>
    <primary>false</primary>
    <values default="" />
    <validator />
 </field>
<field>
    <name>notes</name>
    <description>Some additional notes on the statement.</description>
    <type>TEXT</type>
    <label>Notes</label>
    <sqlFieldName>malcolm_notes</sqlFieldName>
    <sqlCreateStatement>malcolm_notes blob</sqlCreateStatement>
    <namespace>malcolm</namespace>
    <required>false</required>
    <orderBy>false</orderBy>
    <primary>true</primary>
    <values default="" />
    <validator />
 </field>
<field>
    <name>database</name>
    <description>The database containing the statement.</description>
    <type>TEXT</type>
    <label>Database</label>
    <sqlFieldName>malcolm_database</sqlFieldName>
    <sqlCreateStatement>malcolm_database varchar(50)</sqlCreateStatement>
    <namespace>malcolm</namespace>
    <required>false</required>
```

```
            <orderBy>false</orderBy>

            <primary>false</primary>

            <values default="" />

            <validator />

          </field>

        </fields>

      </metadata>
```

Interface XML & XSL

```
<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

              xmlns:malcolm="http://malcolm.xtracker.co.za"

              version="1.0">

  <xsl:output method="html" indent="yes"  version="4" omit-xml-declaration="yes" standalone="yes"/>

  <xsl:template match="interface">

    <xsl:variable name="title" select="meta/field[@id = 'document.title']" />

    <xsl:variable name="identifier" select="meta/field[@id = 'document.identifier']" />

    <xsl:variable name="subject" select="meta/field[@id = 'document.subject']" />

    <xsl:variable name="action" select="meta/field[@id = 'malcolm.action']" />

    <html>

      <head>

        <title><xsl:value-of select="$title" /></title>

        <link rel="stylesheet" type="text/css" href="/styles/default.css" />

        <xsl:apply-templates select="meta" />

        <xsl:comment>

          <xsl:value-of select="components/component[@id = 'rdf']" disable-output-escaping='yes' />

        </xsl:comment>

      </head>

      <body>

        <xsl:apply-templates select="components">

          <xsl:with-param name="title" select="$title" />

        </xsl:apply-templates>

      </body>

    </html>

  </xsl:template>
```

```
<xsl:template match="meta">
    <xsl:for-each select="field">
        <meta name="{@id}" content="{.}" />
    </xsl:for-each>
</xsl:template>
<xsl:template match="components">
    <xsl:param name="title" />
    <xsl:param name="identifier" />
    <xsl:param name="subject" />
    <xsl:param name="action" />
    <xsl:comment>User defined scripts</xsl:comment>
    <xsl:value-of select="component[@id = 'scripts']" disable-output-escaping='yes' />
    <table border='0' width='780' class="main">
        <tr>
            <td width='100%' align='center' class="navigation_top">
                <table border="0" width="100%">
                    <tr>
                        <xsl:variable name="navigation_top" select="component[@id = 'navigation.top']" />
                        <xsl:for-each select="$navigation_top/links/link">
                            <td><xsl:value-of select="." disable-output-escaping='yes' /></td>
                        </xsl:for-each>
                    </tr>
                </table>
            </td>
        </tr>
        <tr>
            <td width='100%' align='center'>
                <table width='100%' align='center' cellpadding='5px'>
                    <tr>
                        <td colspan='3' class="title">
                            <xsl:value-of select="$title" />
                        </td>
                    </tr>
                    <tr>
                        <td width='17%' valign='top' class="navigation_left">
                            <table border="0" class="navigation_left">
                                <xsl:variable name="navigation_left_top" select="component[@id = 'navigation.left.top']" />
                                <xsl:for-each select="$navigation_left_top/links/link">
                                </xsl:for-each>
                            </table>
                        </td>
                        <td width='83%' align='left' valign='top'>
                            <xsl:if test="component[@id = 'navigation.right.top']">
                                <xsl:variable name="navigation_right_top" select="component[@id = 'navigation.right.top']" />
                                <table border='0' align='right' width='22%' class="navigation_right">
                                </table>
                            </xsl:if>
                            <div class="content">
                               <xsl:value-of select="component[@id = 'content.top']" disable-output-escaping='yes' />
                                <xsl:if test="component[@id = 'content.middle']">
                                    <xsl:value-of select="component[@id = 'content.middle']" disable-output-escaping='yes' />
                                </xsl:if>
                                <xsl:if test="component[@id = 'content.bottom']">
                                    <xsl:value-of select="component[@id = 'content.bottom']" disable-output-escaping='yes' />
                                </xsl:if>
                            </div>
                        </td>
                    </tr>
                </table>
            </td>
        </tr>
        <tr>
            <td width='100%' align='center'>
                <xsl:choose>
                    <xsl:when test="component[@id = 'footer']">
                        <xsl:value-of select="component[@id = 'footer']" disable-output-escaping='yes' />
                    </xsl:when>
                    <xsl:otherwise>
                        <table border='0' width='100%' cellpadding='0' cellspacing='0'>
                            <tr>
```

```
                        <td width='57%' align='right' class="footer">
                            (c) 2003 Paul Bothma
                        </td>
                        <td align='right' class="footer">
                            <a href='#top'>top of page</a>
                        </td>
                    </tr>
                </table>
            </xsl:otherwise>
        </xsl:choose>
    </td>
</tr>
</table>
</xsl:template>
</xsl:stylesheet>
```