

Open Digital Libraries

Hussein Suleman

Dissertation submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science and Applications

Edward A. Fox, Chair
James D. Arthur
Roger W. Ehrich
Srinidhi Varadarajan
Gail McMillan

November 19, 2002

Blacksburg, Virginia Tech

Keywords: digital library, interoperability, system architecture, protocol, component,
open archive

© 2002 Hussein Suleman

Open Digital Libraries

Hussein Suleman

(ABSTRACT)

Digital Libraries (DLs) are software systems specifically designed to assist users in information seeking activities. Stemming from the intersection of library sciences and computer networking, traditional DL systems impose library philosophies of structure and management on the sprawling collections of data that are made possible through the Internet.

DLs evolve to keep pace with innovation on the Internet so there is little standardization in the architecture of such systems. However, in attempting to provide users with the highest possible levels of service with the minimum possible effort, many systems work collaboratively with others, e.g., meta-search engines. This type of system interoperability is encouraged by the emergence of simple data transfer protocols such as the Open Archives Initiative's Protocol for Metadata Harvesting (OAI-PMH).

Open Digital Libraries are an extension of the work of the OAI. It is proposed in this dissertation that the philosophy and approach adopted by the OAI can easily be extended to support inter-component interaction within a componentized DL. In particular, DLs can be built by connecting small components that communicate through a family of lightweight protocols, using XML as the data interchange mechanism. In order to test the feasibility of this, a set of protocols was designed based on a generalization of the work of the OAI. Components adhering to these protocols were implemented and integrated into production and research DLs. These systems were then evaluated for simplicity, reusability, and performance.

On the whole, this study has shown promise in the approach of applying the fundamental concepts of the OAI protocol to the task of DL component design and implementation. Further, it has shown the feasibility of building componentized DL systems using techniques that are a precursor to the Web Services approach to system design.

Thanks are given for the support of NSF through its grants CCR-9627922 and IIS-0002935. Thanks also are given to the AmericanSouth.org project funded by the Mellon Foundation.

Acknowledgements

First and foremost, I wish to thank my advisor, Dr Fox, whose constant encouragement, inspiration, and support led me to believe in the importance of my work.

Thanks are due to my family, especially my mother, who from afar served as my conscience by ensuring that working towards finishing my degree was my first priority.

Thanks to the members of the Digital Library Research Laboratory at Virginia Tech, who contributed useful feedback throughout the course of my research. Special thanks go to the late Robert France, as it was during a conversation with him in early 2001 that the initial idea for my research was born.

Without enumerating their names, I acknowledge the support and feedback from many collaborators at Virginia Tech and elsewhere who I have worked with during the development of OAI-PMH and ODL.

Lastly, thanks to my friends all over the world, who have at some time or the other lived with me in person or in spirit, for reaffirming my belief in the fundamental nature of people and for being by my side as I discovered and rediscovered myself.

TABLE OF CONTENTS

LIST OF FIGURES.....	IX
LIST OF TABLES.....	XI
CHAPTER 1 INTRODUCTION.....	1
1.1 PREAMBLE.....	1
1.2 CONTEXT.....	1
1.2.1 <i>What is a Digital Library?</i>	1
1.2.2 <i>What is Interoperability?</i>	2
1.2.3 <i>What is an Open Archive?</i>	2
1.3 MOTIVATION.....	3
1.4 OPEN DIGITAL LIBRARY DESIGN.....	6
1.5 RESEARCH CONTRIBUTIONS.....	9
1.6 OUTLINE OF DISSERTATION.....	9
CHAPTER 2 BACKGROUND: INTEROPERABILITY AND ARCHITECTURE.....	10
2.1 INTRODUCTION TO THE OAI.....	10
2.1.1 <i>Historical Background and Context</i>	10
2.1.2 <i>Initial Technical Efforts</i>	11
2.1.3 <i>Evaluation: Community and Technical Meetings</i>	11
2.1.4 <i>Other Interoperability Efforts</i>	12
2.1.5 <i>DL Architecture Efforts</i>	13
2.2 BASIC OA CONCEPTS.....	14
2.2.1 <i>Repositories and Open Archives</i>	14
2.2.2 <i>Harvesting and Federation</i>	15
2.2.3 <i>Metadata and Data</i>	16
2.2.4 <i>Data and Service Providers</i>	16
2.3 TECHNICAL FRAMEWORK.....	17
2.3.1 <i>Underlying Technology and Standards</i>	17
2.3.2 <i>Sets</i>	18
2.3.3 <i>Records</i>	18
2.3.4 <i>OAI Protocol for Metadata Harvesting</i>	19
2.3.5 <i>Flow Control</i>	20
2.3.6 <i>Registration Services</i>	21
2.3.7 <i>Expansion and Customization</i>	21
2.4 REQUIREMENTS TO BE A PROVIDER.....	21
2.4.1 <i>Data Provider</i>	21
2.4.2 <i>Service Provider</i>	22
2.4.3 <i>Tools and Support</i>	22
2.5 OAI SUPPORT FOR TYPICAL SERVICES.....	23
2.5.1 <i>Cross-Archive Searching</i>	23
2.5.2 <i>Reference Linking</i>	23
2.5.3 <i>Annotations</i>	23
2.5.4 <i>Filtering</i>	23
2.5.5 <i>Browsing</i>	24
2.6 DIGITAL LIBRARY POLICIES FROM AN OAI PERSPECTIVE.....	24
2.6.1 <i>Ownership and dissemination control over digital objects and metadata</i>	24
2.6.2 <i>Changes and withdrawal</i>	24
2.6.3 <i>Preservation</i>	24

2.6.4 Uniqueness of objects and collections	25
2.7 BUILDING OAI SUB-COMMUNITIES	25
2.7.1 Metadata formats	25
2.7.2 Protocol extensions	25
2.7.3 Shared semantics	26
2.8 CASE STUDY: OAI IN THE NDLTD COMMUNITY	26
2.8.1 Context	26
2.8.2 Development of OAI MARC format	26
2.8.3 MetaLibraries	27
2.8.4 Name authority systems	27
2.8.5 Search and Classification for ETDs	28
2.9 FUTURE DIRECTIONS	28
CHAPTER 3 ODL DESIGN CONSIDERATIONS	30
3.1 INTRODUCTION	30
3.2 ODL VS. THE INTERNET: A PRACTICAL PERSPECTIVE	30
3.2.1 Simplicity	32
3.2.2 Openness	32
3.2.3 Independence of protocols	32
3.2.4 Loose coupling	33
3.2.5 Layers	33
3.2.6 Reuse	33
3.2.7 Orthogonality with a Purpose	34
3.3 OPEN DIGITAL LIBRARY DESIGN	34
3.4 ODL VS. OOP: A THEORETICAL PERSPECTIVE	36
3.4.1 Overview	36
3.4.2 Mappings	37
3.4.3 Implications	38
CHAPTER 4 OPEN DIGITAL LIBRARY SERVICE PROTOCOLS	39
4.1 ODL SERVICES AS EXTENSIONS OF THE OAI-PMH	39
4.2 PROTOCOL DESIGN CONSIDERATIONS	39
4.2.1 OAI Sets as Parameters	39
4.2.2 Interface-directed Responses	39
4.2.3 Harvesting Granularity	40
4.2.4 Response-level Containers	40
4.2.5 Submission	40
4.2.6 Harvesting vs. Archive Access	41
4.2.7 Dublin Core Requirement	41
4.2.8 Customization of Components	41
4.2.9 Propagation of Information	41
4.3 EXTENDED OAI-PMH (XOAI-PMH)	41
4.3.1 Global Changes	42
4.3.2 Service Request Changes	42
4.4 PREFACE TO ODL PROTOCOL DESCRIPTIONS	44
4.5 THE ODL-SUBMIT PROTOCOL V1.0	44
4.5.1 Description	44
4.5.2 Interface Protocol	44
4.5.3 Interoperability Issues	45
4.6 THE ODL-RECENT PROTOCOL V1.0	45
4.6.1 Description	45
4.6.2 Interface Protocol	45
4.6.3 Interoperability Issues	45
4.7 THE ODL-UNION PROTOCOL V1.0	46
4.7.1 Description	46
4.7.2 Interface Protocol	46

4.7.3 Interoperability Issues.....	47
4.8 THE ODL-SEARCH PROTOCOL V1.0	48
4.8.1 Description.....	48
4.8.2 Interface Protocol	49
4.8.3 Interoperability Issues.....	50
4.8.4 Query Language: odlsearch1.....	50
4.8.5 Query Language: odlsearch2.....	51
4.9 THE ODL-BROWSE PROTOCOL V1.0	52
4.9.1 Description.....	52
4.9.2 Interface Protocol	52
4.9.3 Interoperability Issues.....	54
4.9.4 Query Language: odlbrowse1.....	54
4.10 THE ODL-RECOMMEND PROTOCOL V1.0	55
4.10.1 Description.....	55
4.10.2 Interface Protocol	55
4.10.3 Interoperability Issues.....	57
4.11 THE ODL-RATE PROTOCOL V1.0	58
4.11.1 Description.....	58
4.11.2 Interface Protocol	58
4.11.3 Interoperability Issues.....	60
4.12 THE ODL-ANNOTATE PROTOCOL V1.0	60
4.12.1 Description.....	60
4.12.2 Interface Protocol	60
4.12.3 Interoperability Issues.....	63
4.12.4 Annotation Metadata Sub-Format: odlannotate1	63
4.12.5 Annotation Metadata Sub-Format: discuss.....	64
4.13 THE ODL-REVIEW PROTOCOL V1.0	64
4.13.1 Description.....	64
4.13.2 Interface Protocol	67
4.13.3 Transaction Formats.....	69
4.13.4 Report Formats	74
4.13.5 Interoperability Issues.....	77
4.14 CASE STUDY: USING AN ODL-SEARCH COMPONENT.....	78
CHAPTER 5 IMPLEMENTATION AND CASE STUDIES	80
5.1 INTRODUCTION	80
5.2 IMPLEMENTATION	81
5.2.1 Platform	81
5.2.2 Customization	81
5.2.3 Component Details.....	82
5.2.4 OAI Component Details.....	93
5.2.5 User Interfaces.....	95
5.3 CASE STUDIES.....	101
5.3.1 Case study: ETD Union Catalog.....	101
5.3.2 Case study: CSTC	103
5.3.3 Case study: husseinsspace.com.....	105
5.3.4 Case study: JERIC	106
5.3.5 Case study: New CSTC	111
5.4 SUMMARY	114
CHAPTER 6 COMPONENT TESTING.....	115
6.1 INTRODUCTION	115
6.2 DIRECT COMPONENT LOGIC	115
6.2.1 IRDB	115
6.2.2 DBBrowse	116
6.3 XOAI INTERFACE	117

6.4 WEB CLIENT TEST	118
6.5 PARSING TEST	119
6.6 REPOSITORY EXPLORER	120
6.6.1 Introduction.....	120
6.6.2 Design of the Repository Explorer.....	121
6.6.3 Validation Procedure.....	123
6.6.4 Options.....	124
6.6.5 Automatic Testing	124
6.6.6 Component Explorer.....	125
6.6.7 Feedback.....	126
CHAPTER 7 ANALYSIS AND EVALUATION	127
7.1 INTRODUCTION	127
7.2 UNDERSTANDABILITY AND SIMPLICITY	127
7.2.1 Methodology	127
7.2.2 Results.....	128
7.2.3 Discussion.....	130
7.2.4 Conclusions.....	132
7.3 REUSABILITY	133
7.3.1 Case study: AmericanSouth.org.....	133
7.3.2 Case study: CITIDEL.....	133
7.3.3 Case study: BICTEL/e.....	134
7.4 EXTENSIBILITY	135
7.4.1 Sub-classing.....	135
7.4.2 Layering.....	135
7.5 PERFORMANCE	136
7.5.1 Communications and Protocol Overhead.....	136
7.5.2 Execution Speed: Nested Components	139
7.5.3 Execution Speed Optimizations.....	141
7.5.4 Load Analysis.....	146
7.5.5 User Interface Response	147
7.5.6 Storage.....	148
7.5.7 Duplication of Data	149
7.5.8 Consistency.....	150
7.5.9 Network Bandwidth.....	151
7.6 SUMMARY	152
CHAPTER 8 FUTURE WORK	153
8.1 INTRODUCTION	153
8.2 HARVESTING	153
8.2.1 Issues with Multiple Sources.....	153
8.2.2 Issues with Single Sources	154
8.3 USER INTERFACES	154
8.3.1 User Interaction API.....	154
8.3.2 MDEdit Generalization.....	155
8.3.3 Component Composition GUI.....	156
8.3.4 Portals.....	156
8.4 LOGGING	157
8.5 SECURITY	158
8.6 NEW PROTOCOLS AND COMPONENTS	158
8.7 TESTING	159
8.8 INSTALLATION AND REGISTRATION.....	160
8.9 PERFORMANCE	161
8.10 NEW STANDARDS	162
8.10.1 OAI-PMH v2.0	162
8.10.2 SOAP-OAI and SOAP-ODL.....	162

8.10.3 ODL v2.0.....	163
CHAPTER 9 CONCLUSIONS	165
REFERENCES	167
APPENDIX A SAMPLE XML SCHEMA FOR MEDIT.....	184
VITA.....	187

LIST OF FIGURES

Figure 1.1 Internal structure of typical ODL search component	7
Figure 1.2 Initial architecture of the CITIDEL system.....	8
Figure 1.3 Architecture of the NDLTD Union Catalog Experimental System.....	8
Figure 2.1 Data flow for federation and harvesting.....	15
Figure 2.2 Layered organization of data storage and service provision	16
Figure 2.3 Fragment of XML illustrating namespaces and schema locations.....	18
Figure 2.4 Sample record from the arXiv open archive.....	19
Figure 2.5 Minimal metadata record from arXiv with optional <about> section.....	21
Figure 2.6 Example sequence of requests and responses between service and data providers	22
Figure 2.7 Fragment of sample record of XML encoding of MARC.....	27
Figure 3.1 Encapsulation of network protocols, up to the level of ODL.....	31
Figure 3.2 Example networked architecture of an Open Digital Library	36
Figure 4.1 Simple ODL network using ODL-Union-compliant component	46
Figure 4.2 Hierarchical organization of ODL-Union-compliant components.....	48
Figure 4.3 Simple ODL network using an ODL-Search-compliant component	49
Figure 4.4 Interface and component interaction during indexing and search operations .	79
Figure 5.1 Directory layout for a typical component.....	82
Figure 5.2 Internal architecture of DBUnion.....	83
Figure 5.3 Sample configuration for DBUnion	84
Figure 5.4 Internal architecture of IRDB.....	86
Figure 5.5 Fragment of DBBrowse configuration	87
Figure 5.6 Direct editing interface to Box component	88
Figure 5.7 Insertion procedure for new entries in a Thread component.....	90
Figure 5.8 Sample configuration script for Filter	94
Figure 5.9 HTML rendering of metadata editor	100
Figure 5.10 Architecture of NDLTD ODL system.....	102
Figure 5.11 Index page for NDLTD user interface.....	102
Figure 5.12 Output from typical browse operation.....	103
Figure 5.13 Architecture of CSTC, showing ODL components.....	104
Figure 5.14 CSTC interface for browsing, using DBBrowse component	104
Figure 5.15 Architecture of the guestbook addition to husseinsspace.com.....	105
Figure 5.16 User interface for the guestbook on husseinsspace.com	106
Figure 5.17 Architecture of JERIC peer review system	109
Figure 5.18 User interface of JERIC peer review system.....	110
Figure 5.19 Full details for a single resource or section	111
Figure 5.20 Architecture of new CSTC system	112
Figure 5.21 New CSTC initial welcome screen.....	112
Figure 5.22 New CSTC list of editing, reviewing, and submission tasks	113
Figure 5.23 New CSTC resource browsing	113
Figure 5.24 New CSTC full metadata and associated information	114
Figure 6.1 Layers at which component testing can be performed	115

Figure 6.2 Internet Explorer displaying an ODL request and response.....	120
Figure 6.3 Repository Explorer basic interface	121
Figure 6.4 Repository Explorer verb and parameter entry	122
Figure 6.5 Hyperlinked response from ListSets	122
Figure 6.6 Flow of data during validation/testing process.....	123
Figure 6.7 Language selection in Repository Explorer	124
Figure 6.8 Front page listing of components in Component Explorer	126
Figure 7.1 Architecture of simple componentized digital library.....	127
Figure 7.2 Original architecture of CITIDEL, showing metadata layer distinct from service layer	134
Figure 7.3 Testable interfaces for IRDB component.....	137
Figure 7.4 Load conditions with SpeedyCGI and without	147
Figure 8.1 Fragment of DBBrowse log file	157
Figure 8.2 Example of SOAP GetRecord service request body	163
Figure 8.3 Current and proposed models for OAI/ODL relationship.....	164

LIST OF TABLES

Table 2.1 Service requests in the OAI Protocol for Metadata Harvesting.....	20
Table 3.1 Mapping of concepts from ODL \leftrightarrow OOP	36
Table 4.1 Assignment control matrix.....	66
Table 5.1 ODL reference components, descriptions, and protocols	80
Table 5.2 Names and descriptions of OAI components	81
Table 5.3 Parameters for DBUnion	85
Table 5.4 Parameters for DBBrowse	87
Table 5.5 Parameters for Filter	95
Table 5.6 List of additional MDEdit schema tags to define appearance of HTML forms	99
Table 7.1 Results to background information questions on survey	129
Table 7.2 Responses to questions on exercise	130
Table 7.3 Execution times for request submitted to different layers of ODL-IRDB.....	138
Table 7.4 Number of matches for each query	138
Table 7.5 Time differences between pairs of consecutive tests.....	138
Table 7.6 Response times for ListIdentifiers vs. ListRecords	140
Table 7.7 Regular CGI vs. SpeedyCGI speed comparisons	142
Table 7.8 SpeedyCGI vs. direct API speed comparisons	142
Table 7.9 Average execution times under different load conditions	146
Table 7.10 Average execution times when using SpeedyCGI.....	146
Table 7.11 Execution times for user interface actions.....	148
Table 7.12 Illustration of duplication due to overlapping.....	150

Chapter 1

INTRODUCTION

1.1 PREAMBLE

“What on earth is 'digital libraries' anyway?”

- A. Rakgole, Personal communication, May 2001

At first it was startling that people respond in this manner but that concern has been replaced by a reluctant acceptance that most people, even fellow computer programmers, do not have any idea what a Digital Library is. While to some it is obvious because of personal experience, most people know either too little or too much to be able to concisely define a Digital Library.

It is hardly the fault of computer users since the field is far from being well defined. Unlike word processing or web browsing, where most computer users can cite at least one example of a software package, Digital Libraries do not conjure up any immediate images of familiarity. Instead, the ground has yet to be trod and systems research, as opposed to systems production, is still largely the rule rather than the exception.

Recently much of this research has started to focus on the issues that deal with connecting together distributed computer systems for information transfer instead of the traditional distributed computing use. In this context, with the creation of new protocols and the adoption of new software design models, it may be possible to design a new breed of Digital Libraries that both support ongoing research and rapid installation of production systems. That is the ultimate aim of this research, as motivated and expanded upon in the following sections.

1.2 CONTEXT

Before launching into a detailed analysis of the problem space and the solution suggested by this work, it is essential to set a proper context by defining the axial concepts of this study.

1.2.1 What is a Digital Library?

A Digital Library is an electronic information storage system focused on meeting the information seeking needs of its users.

There are many definitions for Digital Libraries (DLs) used in practice, each of which attempts to model different facets of existing systems in order to create a well-defined subspace of the set of all computer systems. Some definitions choose to enumerate the

services offered by DLs, such as searching and browsing, as being necessary and/or sufficient to distinguish a DL from a non-DL. Other definitions try to fit DLs into formal frameworks that may support a rigorous mathematical model. On a different plane, some definitions will place emphasis on the human-centric aspects while others may stress the machine-oriented aspects.

Consider several exemplary definitions. The Digital Library Federation (CLIR, 1998) takes a typical approach by covering as many of the different aspects as possible while Levy and Marshall choose a more minimalist approach (Levy and Marshall, 1995). Lesk takes the extremely simple yet effective approach of stating that “you need to get stuff into it, you need to be able to get stuff out of it...” (Lesk, 1997).

For the purposes of this study, a minimalist definition, such as what is provided in the opening paragraph, is sufficient.

Examples of popular digital libraries are HowStuffWorks (Brain, 2002), Los Alamos Physics e-Print archive (LANL, 2002), Internet Movie Database (Amazon.com, 2002), Library of Congress’ American Memory project (Library of Congress, 2002a), and the Networked Computer Science Technical Reference Library (Davis and Lagoze, 2000).

1.2.2 What is Interoperability?

Merriam-Webster (Merriam-Webster, 2002) defines *interoperability* as follows:

“Ability of a system (as a weapons system) to use the parts or equipment of another system”

In the DL field, this definition contains appropriate analogies to respective computer systems that share data and systems that work cooperatively in order to provide services to users. An example of the former would be when databases are merged to form a central searchable collection of data, like the OCLC WorldCat collection (OCLC Inc., 2002b). An example of the latter would be when a meta-search engine uses multiple search engines in its strategy to provide a more complete service for its users, like the MetaCrawler system (InfoSpace, 2002).

In general, interoperability refers to the ability of a DL to work cooperatively with other DLs in an attempt to provide higher quality services to users. There are many approaches to achieve some degree of interoperability and one such approach involves the creation and use of Open Archives.

1.2.3 What is an Open Archive?

An Open Archive is a computer interface to access a collection of data, where the interface conforms to the specifications laid down by the Open Archives Initiative’s (OAI, 2002) Protocol for Metadata Harvesting (Lagoze, et al., 2002).

The Open Archives Initiative is an organization formed by a broad range of researchers, librarians, publishers, and archivists whose aim is to create simple standards to support interoperability among systems. The most recent standard is the Protocol for Metadata

Harvesting, which specifies how two computer systems may communicate a stream of structured records from one to the other on a continuous basis. A system that contains the source data and that conforms to this protocol is called an Open Archive.

Examples of some DL systems that are, in whole or part, Open Archives are: Computer Science Teaching Center (Fox, et al., 2002b), Computing and Information Technology Interactive Digital Educational Library (Fox, et al., 2002a), Los Alamos e-Print archive (LANL, 2002), the American Memory Project (Library of Congress, 2002a), and the National Science, Technology, Engineering, and Mathematics Education Digital Library (NSF, 2002).

1.3 MOTIVATION

Given the loosely defined nature of Digital Libraries it is hardly surprising that the field does not easily converge on standards and technology. Most of the existing systems that are classified as DLs are the result of custom-built software development projects with intensive design, implementation and testing cycles. There are many reasons why this effort is repeated for each project:

- ◆ Many DLs are built in isolation as a response to the needs of a particular community, in most cases not involving personnel with prior experience.
- ◆ Most modern DLs have WWW interfaces – thus the user interfaces and process flows are fashioned to resemble the way people use the WWW, which itself changes with time.
- ◆ Each DL is aimed at meeting the needs of a particular community – so the underlying program logic varies vastly among systems.
- ◆ Most DLs are intended to be quick solutions to urgent community needs – so not much thought goes into planning for future redeployment of the systems.
- ◆ DLs, by the very nature of being responses to user needs, can be arbitrarily complex so new projects sometimes choose to develop from scratch because it is cheaper than adapting what already exists to a different scenario. As testimony to this, at the turn of the millenium, Dijkstra wrote that computing's central challenge of “how not to make a mess of it” had not been met (Dijkstra, 2001).
- ◆ As DL systems get more complex, extensibility becomes more difficult and as a result maintainability is compromised.
- ◆ There are very few software toolkits available to build DLs.

The natural solution would be to create software toolkits. A few institutions have investigated that approach. Dienst (Lagoze and Davis, 1995) is a DL system developed at Cornell University with tasks clearly divided and specified by a protocol based on HTTP and eventually using XML. It was developed to support the distributed operation of the NCSTRL project and, while technically sound, required an investment in software, methodology, and support that some prospective users were not willing to make. The

Repository-in-a-Box (NHSE, 2002) software from University of Tennessee is an alternative as is the E-Prints (OpCit, 2002) software from Southampton University. Both these toolkits avoid many problems related to complexity of DLs by defining workflows that are not easy to change. All of these and other systems have had varying degrees of success among archivists looking for drop-in solutions but they generally suffer from two basic problems:

- ◆ The range of possible workflows is restricted by the design of the system.
- ◆ The software is either built as a monolithic system or as components that communicate using non-standard protocols – in both cases making understanding and modification a complex process.

Most modern programming environments adopt some form of component model as it is widely accepted as good software engineering practice. Even in the DL community, as far back as 1994, early discussions on the future of DLs (Gladney, et al., 1994) concluded that components were an integral part of the solution. Other scientific communities embraced component technology as an aid to rapidly and correctly solving problems - for example, the Sieve framework at Virginia Tech (Sieve, 2002) encapsulates scientific functionality into software components. However, in spite of the widespread use of such technology, for the reasons outlined above, the DL community did not in general adopt a single component framework.

In October of 1999 the Open Archives Initiative (OAI) (Van de Sompel and Lagoze, 2000) was launched in an attempt to address interoperability issues among the many existing and independent DLs. The focus was on high-level communication among systems and simplicity of protocols. The OAI has since received much media attention in the DL community and, primarily because of the simplicity of its standards, has attracted many early adopters.

The OAI Protocol for Metadata Harvesting in essence supports a system of interconnected components, where each component is a DL. Also, since the protocol is simple and widely accepted it is far from being a custom solution of a single project. The OAI protocol can be thought of as the glue that binds together components of a larger DL. However, since DLs are themselves defined only loosely, this collaborative system can be composed of individual component DLs, each with different functionality. In the extreme case, each component DL can supply the functionality of exactly one service expected by a user (or part of a service). This leads to the basic hypothesis of this research:

*Digital Libraries can be modeled as networks of extended Open Archives,
where each extended Open Archive is a source of data and/or a provider
of services.*

The “extensions” are necessary since Open Archives are optimized for the provision of data – but are generalizable to other tasks with a few minor changes.

This hypothesis is further motivated by the following factors:

- ◆ Componentization is built into the system by design if every service is delivered by an extended Open Archive. This inherently supports reuse and allows for interoperability at the level of individual services within the DL.
- ◆ It more closely resembles the way that physical libraries work. In a physical library the individual systems interoperate within their own communities. For example, the purchasing department interoperates with the booksellers and the inter-library loan department interoperates with peer departments at other libraries. The head of the acquisitions department can be replaced because he or she understands a common protocol for all libraries. Interoperability is achieved at the level of individual services rather than at the level of organizations.
- ◆ There is currently a significant difference in technology between research DLs and production DLs. The former focuses on experimental concepts and technology while the latter deals with the real issues of meeting the needs of users. Connecting the two is not usually a simple task, but if both systems subscribed to a common protocol that will greatly simplify matters – OAI can supply that protocol.
- ◆ The Internet is without a doubt the single most effective information dissemination tool of current times. This was primarily possible because of the simplicity of the protocols it relied on and the hierarchical manner in which protocols such as HTTP (Fielding, et al., 1999) built on more fundamental protocols such as IP and TCP. The OAI provides us with a simple protocol to transfer metadata; building simple layered extensions to this protocol will closely follow the proven methodology of the networking community (ISO, 1994).
- ◆ While complex system interactions might support complex operations, they also raise the bar on adoption of new technology. A good example is the hypertext community where the WWW has succeeded well beyond other projects simply because its model was always a simple one (Berners-Lee and Fischetti, 1999). Modeling DL services as Open Archives will encourage such a degree of simplicity.
- ◆ Scholarly communication is a rapidly changing field and many people are slow in making the transition to new forms of communication in spite of a growing number of advocates like Stevan Harnad (Harnad, 1999). The success of new DL systems in this arena relies on keeping pace with current thinking on how publications are created, processed, and distributed. A simple component model will greatly simplify changes in the workflow of the DL to support the gradual shift to new and improved processes.
- ◆ User interface design and workflow management are complex tasks but common base-level services – mediators in DL language or middleware in three-tier client-server development (Umar, 1997) – have emerged in practice, for example, searching and browsing. If an arbitrarily complex user interface can access DL components in a

standard manner it will be easier to interchange components and add new services – the OAI protocol can be the basis of that standard protocol for components. Norman advocates that designs should be visible, understandable and natural in their mappings (Norman, 1990). The OAI protocol is already establishing itself in those areas so it makes an ideal foundation upon which to build.

To prove the stated hypothesis a DL component framework based on the OAI protocol was designed, implemented, and tested. The aspects that were tested and analyzed include:

- that the design and implementation are effective in meeting real users' needs,
- that the DL component network model is equivalent in functionality to a monolithic system,
- that there is no significant performance degradation inherent in using a DL component network instead of a monolithic system,
- that the component framework is easily understood and extensible,
- that most DL services can be modeled within this framework, and
- that basic assumptions about the OAI protocol are true, including that it is efficient in transferring metadata and that if used in specific configurations it will correctly and completely duplicate a stream of metadata records.

1.4 OPEN DIGITAL LIBRARY DESIGN

In order to comply with the requirements listed above, the proposed Open Digital Library must be based on the following fundamental design principles:

- All services must be standard components encapsulated within extended Open Archives, accessed by other services through their OAI interfaces, and with all input parameters being instantiations of standard OAI parameters, with semantics overloaded as necessary.
- User interfaces must be custom-built for each application, communicating with a collection of service components through their OAI interfaces for data access (e.g., search) and other extended OAI interfaces for data processing (e.g., review).

Figure 1.1 shows a simplified view of a typical search engine component.

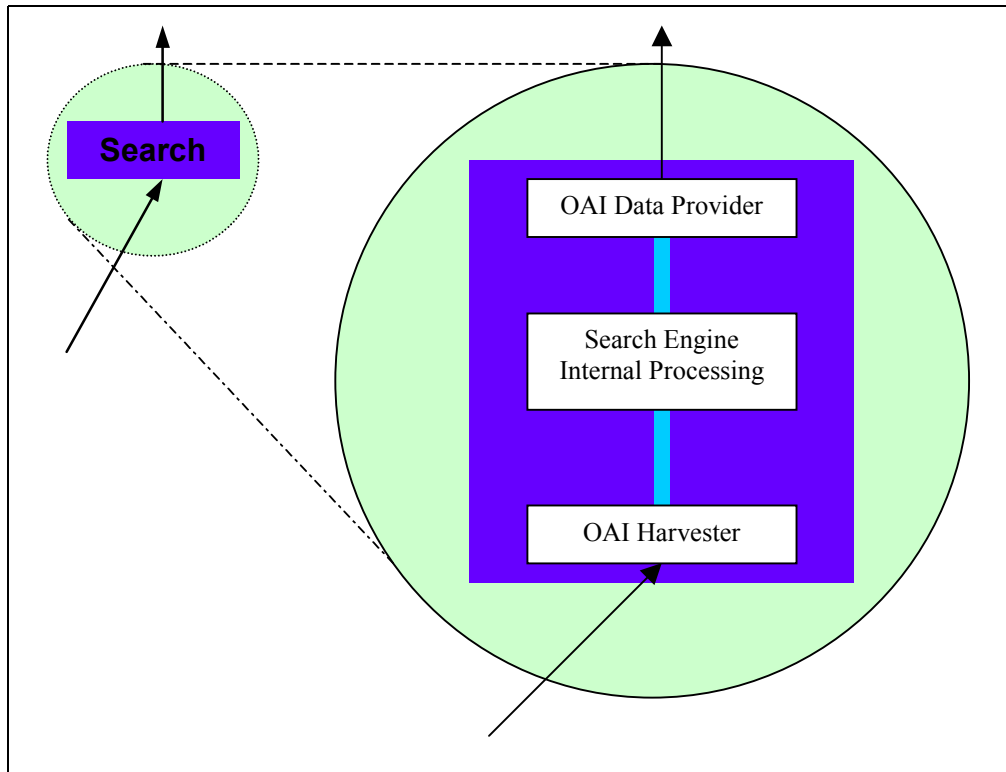


Figure 1.1 Internal structure of typical ODL search component

The reason for standardization in services but not user interfaces is to avoid addressing the arbitrary interface complexity of an online system. At this time, it is not possible to specify a system workflow and user interface in a standard manner without either resorting to the complexity of a programming language or restricting the problem domain. To make an analogy, this research deals with what's under the hood – not the knobs on the dashboard.

Figure 1.2 illustrates the initial design philosophy behind the technical infrastructure for the CITIDEL project. The system is a combination of six Open Archives working cooperatively to provide the functionality of a single DL, complete with support for extended interoperability with other systems. In this case the DL Core is the mechanism by which each of the top-level services accesses each of the bottom-level services, possibly a registry of basic archives.

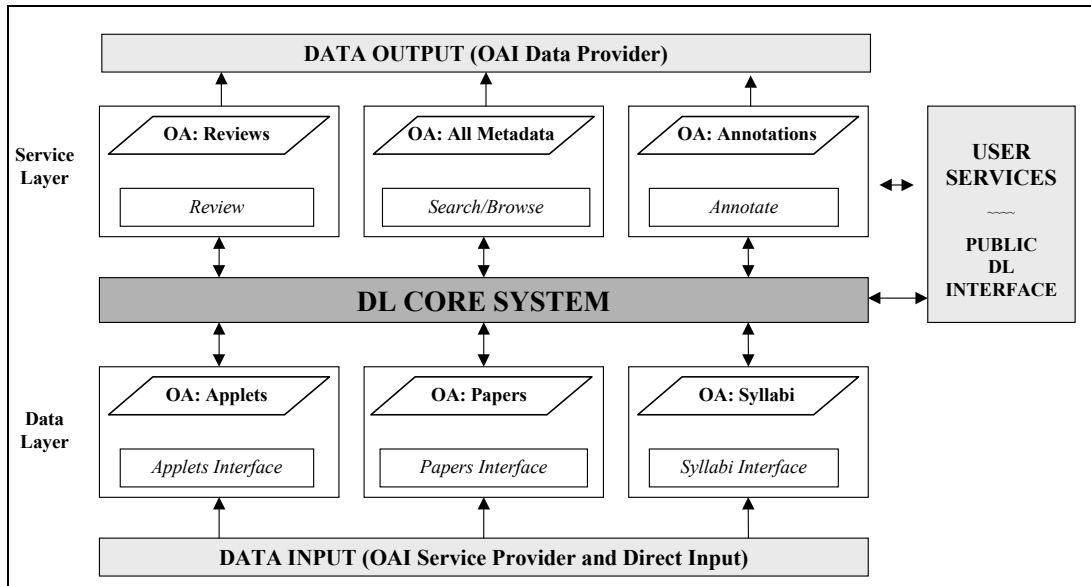


Figure 1.2 Initial architecture of the CITIDEL system

In a different context, the NDLTD Union Catalog project (Suleman, et al., 2001) collects the same metadata from multiple partners and has a different internal structure but utilizes the same basic components as shown in the CITIDEL design. Figure 1.3 shows where OAI is used as a communications medium in the prototype user interface layered over the NDLTD Union Archive or Merged Collection. All arrows represent data being accessed or transferred through OAI or extended OAI interfaces, and all nodes labeled “OA” are Open Archives. The functionality of the “Search”, “Browse” and “Recent” service components is discussed in Chapters 4 and 5.

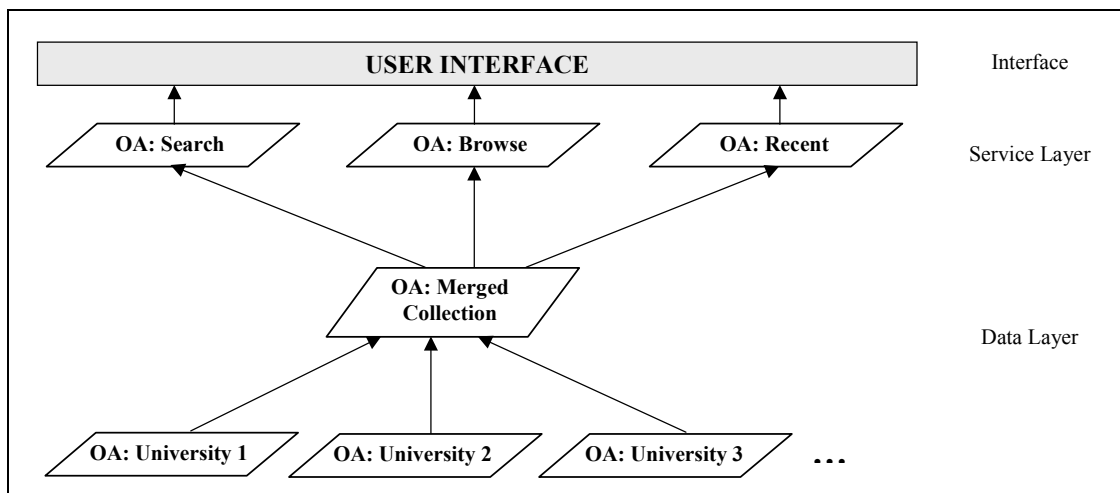


Figure 1.3 Architecture of the NDLTD Union Catalog Experimental System

In a similar fashion, other DL systems can be modeled within this framework, the details of which are outlined and discussed in later sections.

1.5 RESEARCH CONTRIBUTIONS

This research makes the following contributions to the field of DL interoperability and architecture:

- The design of a component framework for DLs based on OAI.
- For each DL service, a set of specific semantics for the general parts of the OAI protocol.
- An evaluation of the Open DL design to show that it meets its stated requirements.
- A verification of the applicability of the standard OAI protocol to DLs.
- Recommendations for extensions of the OAI protocol to support better componentization and interoperability and more functionality within a DL.
- Recommendations for additional/alternative OAI protocols and protocol bindings where applicable to complete the DL framework by complementing what has already been specified.
- A toolkit for building basic componentized ODLs.
- A set of testing tools to validate the OAI protocol and derivatives defined as ODL protocols.

1.6 OUTLINE OF DISSERTATION

Chapter 1 outlines the motivation, problem space, and scope of the research.

Chapter 2 discusses historical and current research in interoperability and architecture, including an in-depth description and analysis of the OAI protocol.

Chapter 3 presents the ODL design framework with parallels with design philosophies drawn from the development of the Internet, and contrasted with Object Oriented Programming.

Chapter 4 defines a comprehensive set of DL services within this design framework, as semantic overlays over the OAI protocol.

Chapter 5 discusses the implementation of this framework in the context of reference components and case studies including the NDLTD Union Catalog, Journal on Educational Resources in Computing (JERIC), and CSTC.

Chapter 6 discusses testing techniques for ODL component implementations to verify both the component logic and external interfaces.

Chapter 7 is an evaluation and analysis of the design and implementation.

Chapter 8 presents recommendations for future work.

Chapter 9 is the conclusion.

Chapter 2

BACKGROUND: INTEROPERABILITY AND ARCHITECTURE

2.1 INTRODUCTION TO THE OAI

2.1.1 Historical Background and Context

The World Wide Web (WWW) is frequently thought of as the technology that revolutionized computer networking by effectively breaking down the barriers between the providers of content and the users of that content. The underlying idea was not particularly a novel one since the hypertext community has been investigating such avenues for decades. However, it was backed up by free, easy to utilize software that satisfied a need in the rapidly advancing networked community, and so it was immensely successful.

The WWW broke down a major barrier in making information freely accessible, but it also created information management problems for which simple solutions did not exist. One such problem is that of persistence: how can we guarantee that a digital object on the WWW will always exist? Another question has to do with authority: how much trust can we place in the authenticity of a source of digital objects? These and other concerns led some individuals and organizations to begin creating managed repositories of digital information, nowadays called Digital Libraries (DLs), with additional and specialized services to enhance the users' experience beyond what the WWW had to offer.

While the WWW thrived because of its distributed nature, most DLs tried to provide one-stop shopping for users in specific communities. As the number of DLs increased, users looking for resources found that they needed to search through many DLs before finding what they needed. Most DLs are driven by databases; thus the popular search engines do not index their contents. As a result, search engines are not of much use to users who want to perform searches across multiple DLs.

In order to address this need, different approaches were taken by various communities of users. The Z39.50 (ANSI/NISO, 1995) protocol was designed for client/server access and adapted to federated searching, whereby a system performing a search operation on multiple repositories could send the query to all of them in a standardized format and then process the returned results as appropriate. The Harvest system (Bowman et al., 1995) attempted to gather metadata from websites and create a central searchable index. The Dienst protocol from Cornell University (Davis and Lagoze, 2000) and the STARTS protocol from Stanford University (Gravano et al., 1997) both implemented variations of federated search algorithms, where queries are sent to remote sites in real-time. Kahn

and Wilensky's Repository Access Protocol (Kahn and Wilensky, 1995) allowed remote access to the contents of a repository, thus facilitating search and browsing operations. These projects had varying degrees of success, in most cases limited to large or research DLs where there was a commitment to building interoperability into the systems. Smaller DLs were not prepared to make the investment in a complex protocol for interoperability, especially since the rewards were not immediately tangible.

In October 1999, a meeting of representatives of various existing archives was held in Santa Fe, New Mexico, USA, to address the concern that interoperability was beyond the reach of most DL systems. Delegates at this meeting included representatives of the Association of Research Libraries, Coalition for Networked Information, Council on Library and Information Resources, Digital Library Federation, Library of Congress, Networked Digital Library of Theses and Dissertations (NDLTD), Scholarly Publishing and Academic Resources Coalition, and various universities and research institutes. The primary focus of delegates was on facilitating the creation of a Universal Preprint Service (Van de Sompel, et al., 2000) – a DL that contained all electronic pre-prints such as papers, articles, and theses. The result of this meeting, the Santa Fe Convention (Van de Sompel and Lagoze, 2000), was an agreement among the parties to subscribe to a common standard for interoperability based on transfer of metadata from repositories using a minimal protocol and leveraging existing technology to achieve this.

2.1.2 Initial Technical Efforts

The Santa Fe Convention laid the groundwork for future efforts by defining the guiding principles of the Open Archives Initiative (OAI) (OAI, 2001) – principles that are largely unchanged after 3 years of further discussion within an expanding community of digital librarians and users of information.

At the Santa Fe meeting, it was decided that archives should be able to exchange metadata with one another using a modified subset of the Dienst protocol. As is often the case, however, this first iteration of the interoperability protocol led to much debate over semantics and ambiguities inherent within the specifications. Early implementations for the Computer Science Teaching Center (Fox, et al., 2002b) and the Physics Preprint Archive (LANL, 2002) were based on subtly different interpretations of the protocol. Discussions among implementers of the protocol convinced some proponents of the Santa Fe Convention that more work was needed to make the protocol specification robust and thus truly standardized. This notion was formalized at two workshops and a technical committee meeting, which, along with a Steering Committee, guided the evolution of that initial protocol into its subsequent incarnations.

2.1.3 Evaluation: Community and Technical Meetings

The OAI held two workshops in conjunction with the ACM DL2000 (San Antonio, USA, June 2000) and ECDL 2000 (Lisbon, Portugal, September 2000) conferences, where the initial work was evaluated and a future course was charted for the OAI.

Unlike the inaugural meeting, these workshops were openly advertised to digital library practitioners and they drew a broad range of participants from sectors of the community

ranging from publishers to researchers. It was unanimously agreed that the initial protocol needed revision and that the OAI needed to broaden its scope to serve communities beyond its initial mandate of pre-print archives. To address these issues, a technical committee was formed and tasked with revising the protocol to eliminate the shortcomings that were recognized and to meet the needs of the larger OAI community. This committee met in September 2000 in Ithaca, NY, USA to launch an intensive period of writing, implementing and testing, which culminated in the official release of version 1.0 of the OAI Protocol for Metadata Harvesting (OAI-PMH) in January 2001 (Lagoze and Van de Sompel, 2001). This protocol, having undergone extensive alpha testing prior to release, promised to provide a simple mechanism for DLs to interoperate effectively. The release of the protocol was, however, underscored by the stipulation that the first version was for experimentation and not stable production-level use.

Later that year the protocol specification received a minor upgrade to version 1.1 because of changes made to the underlying XML Schema standard (Fallside, 2001).

Thereafter, a second technical committee was formed to exhaustively evaluate the initial OAI protocol and design a version to address all of the concerns of the OAI community at large. A few changes were made to the specification to enhance generality while retaining simplicity. (These are discussed in later sections.) After a second lengthy process of alpha and beta testing, the OAI-PMH version 2.0 was released in June 2002 (Lagoze, et al., 2002).

2.1.4 Other Interoperability Efforts

In the meanwhile, various other communities had begun to investigate aspects of this problem and devise solutions that are often contrasted with the approaches taken by the OAI.

In the Business-to-Business (B2B) community, the RosettaNet's Partner Interface Process (Greef, 1998) has attempted to define clear semantics for the interactions among businesses in an attempt to extend the functionality of Electronic Data Interchange (EDI) into the new era of XML-based data exchange. The ebXML project (Grangard, 2001) has gone a step further in trying to define frameworks and registration services to build upon the emerging encoding and discovery standards on the Internet.

As a collaborative effort among industry and academia, a number of protocols based on XML are beginning to emerge. The Simple Object Access Protocol or SOAP (Box, et. al., 2000) defines a standard encapsulation mechanism to support distributed computation through the exchange of XML-encoded data. It does not define the semantics of tasks but merely provides a standard mechanism to specify remote procedure calls and their results. This is something like the XML-RPC protocol (Winer, 1999), and does as well, but XML-RPC is significantly simpler than the more general SOAP protocol. The Web Distributed Data Exchange or WDDX protocol (Simeonov, 1998) is yet another attempt to pass data over the WWW, but with an emphasis on simple structured data as opposed to procedures in XML-RPC and objects in SOAP.

In order to adopt the structured data or remote Web service invocation functionality, it is necessary to first locate the provider of the services and determine the format of requests and responses. To address the latter, the Web Services Description Language or WSDL (Ogbuji, 2000) was created to concisely describe the locations of services along with their parameters and expected results. To address the former issue of service discovery, the Universal Description, Discovery, and Integration of Business for the Web service – or UDDI (Ariba, et al., 2000) – was introduced to provide a distributed registry of Web Services.

In various combinations, these standards promise to change the way that the Web currently operates by allowing for greater automation and interoperability among diverse networked systems. There is one thread that is common to many of these projects, and that is that they are strictly syntactical. While the OAI protocol tries to define a simple syntax and concentrate on the semantics of interoperability, many other widely touted solutions standardize on the syntax but leave semantics to communities. The OAI-PMH, serving the DL community, is in a unique position where it defines the semantics needed to provide access to data sources, and possibly also acts as a layer upon which higher-level DL services may be built.

2.1.5 DL Architecture Efforts

While interoperability is the focus of many current efforts, it is a need that is largely driven by the widely varying designs of existing digital libraries. The DL field has historically been faced with the almost contradictory requirements to build extensible but tightly integrated systems. Users in a particular community expect a search engine to understand them intimately but at the same time developers are expected to surgically extract a search engine and install it into a completely different application. To address this, various projects have attempted to model DLs in ways that integrate interoperability with architecture design.

The Dienst system (Lagoze and Davis, 1995) discussed previously is an implementation of the Dienst protocol – a formal specification of the way in which various components interacting within the digital library use HTTP (and eventually XML as well) as the underlying layer. Members of the Networked Computer Science Technical Reference Library (Leiner, 1998) used earlier versions of Dienst for many years as the basis of their DL architecture and interoperability solution. The Dienst system had varying degrees of success and was a leading influence on the development of the OAI protocol, which now uses newer standards and a more general approach than Dienst. The Open Digital Library (ODL) components build upon this generality to perform innovative functions.

The FEDORA project (Payette and Lagoze, 1998) further developed the Dienst repository architecture by defining abstract interfaces to structured digital objects, initially implemented over a CORBA communications medium.

Also using CORBA is the Stanford InfoBus project (Baldonado, et al., 1997; Roscheisen, et al., 1998), which developed an approach for interconnecting systems using distinct protocols for each purpose. The InfoBus project views interoperable components as an enabler for a suite of higher-level DL services. In contrast, the ODL project uses

interoperable protocols at all levels within and at the exterior edges of componentized systems.

In the field of software agents, multi-agent systems can be thought of as being analogous to distributed digital libraries, since both use a network of intelligent service-based systems to satisfy the information needs of a user. In the case of the DL, this intelligence is implicit in the semantics built into its services. Intelligent agents, on the other hand, are plagued with the age-old problem of achieving far greater success at the level of syntactic interoperability than semantic interoperability (Nwana and Ndumu, 1999). Projects such as the Knowledge Query and Manipulation Language (KQML) (Finin, et al., 1997) address this problem by attempting to define basic semantics for query answering. The University of Michigan Digital Library Project (Birmingham, 1995) built DLs as collections of autonomous agents, with protocol-level negotiation to perform tasks collaboratively. These projects are complicated by the various different layers of definition that are needed – a problem avoided in ODL by building upon the strong foundations already established by the OAI.

The OpenDLib project (Castelli and Pagano, 2002) has taken an approach informed by the development of the OAI-PMH to build DLs as componentized systems with open inter-component communication protocols. While very similar, the prime difference between ODL and OpenDLib is that ODL assumes all components are OA-like to exploit the simplicity and understandability of OAI-PMH.

All of these component models are built upon popular syntactic layers, such as HTTP and CORBA, and define additional semantics where necessary. This need for a common communications mechanism also is a driving force behind interoperability protocols such as SDLIP (Paepcke, 2000) and OAI-PMH. The latter was investigated through ODL as the basis for an alternative glue to bind together components in a DL.

In general, the Open Digital Library is an attempt to infuse interoperability into all aspects of the digital library. Building upon a simple and easily understood protocol like the OAI-PMH has the desirable effect of extending a simple model of semantics into all interactions. Ultimately, it is hoped that such component network approaches to DLs will garner as much support among DL architects as OAI does in the interoperability sector.

2.2 BASIC OA CONCEPTS

2.2.1 Repositories and Open Archives

The words “Open Archive” frequently conjure up images of information access without any associated cost or restriction. While this is a goal for many proponents of the OAI, it places too many restrictions on DLs that want to conform to OAI standards. So, the OAI defines an Open Archive (OA) simply as being an archive that implements the OAI Protocol for Metadata Harvesting, thus allowing remote archives to access its metadata using an “open” standard.

A “Repository” is often used as a synonym for an OA. In the traditional DL context, a repository is a collection of digital objects, but in the context of the OAI, it has to be network accessible and it has to support the OAI Protocol for Metadata Harvesting.

2.2.2 Harvesting and Federation

The first crucial decision made by the OAI was the selection of a method to achieve basic interoperability among repositories, with special emphasis placed on the ability to do cross-archival searching. It is generally considered that there are two major approaches to accomplish this: harvesting and federation.

Federation refers to the case where the DL sends the search criteria to multiple remote repositories and the results are gathered, combined, and presented to the user. Harvesting is when the DL collects metadata from remote repositories, stores it locally and then performs searches on the local copy of the metadata. Figure 2.1 illustrates the differences in data flow.

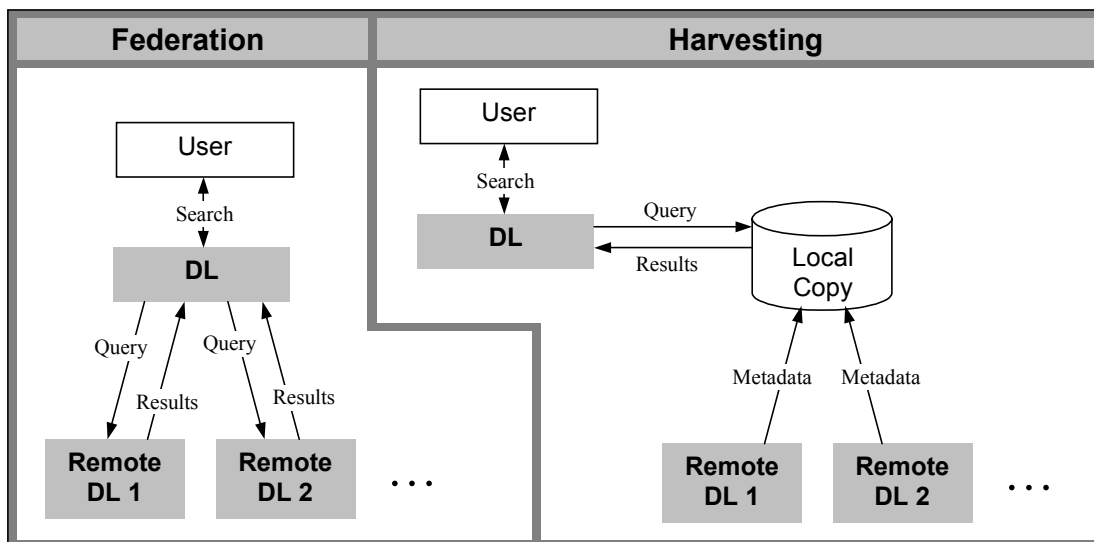


Figure 2.1 Data flow for federation and harvesting

Federation is a more expensive mode of operation in terms of network and search system constraints since each repository has to support a complex search language and fast real-time responses to queries. Harvesting requires only that individual archives be able to transfer metadata to a requesting DL. The frequency of queries, quantity of metadata, and availability of network resources also factor into this comparison but, in general, federation places a greater burden on the remote sites while harvesting reduces the demand on remote sites and concentrates the processing at a central DL site. Since it is likely that providers of services, such as search engines, will expend the effort to store, index, classify, and otherwise manage searchable metadata, the OAI opted for harvesting, primarily as a means of lowering the barrier to interoperability for providers of data.

2.2.3 Metadata and Data

The question of what to harvest is a contentious issue for many, as it is not obvious whether an archive should be sharing its metadata, its digital objects, or both. There are advantages to exchanging complete digital objects since that will support operations like full-text search of text documents. However, in most instances DLs need only harvest metadata in order to provide search, classification, and related services. This approach was adopted by the OAI, with the implicit understanding that the metadata will contain pointers to the concrete rendering of digital objects.

2.2.4 Data and Service Providers

A data provider maintains a repository that allows external online access to its metadata through the OAI Protocol for Metadata Harvesting. In the interest of brevity, “data provider” is sometimes used to refer to such repositories. A service provider is an entity that harvests metadata from data providers in order to present users with higher-level services. This distinction allows for a clean separation between the provider of data and the provider of services (as illustrated in Figure 2.2). This helps eliminate the current barrier to quality services that arose because of the historical connection between ownership of data and provision of services. In general, archives with large quantities of content emphasize information management over the provision of user services. On the other hand, if information management is not a primary function of an archive, more effort can be devoted to service provision. The OAI attempts to clarify and separate these approaches to present users with the best of both worlds.

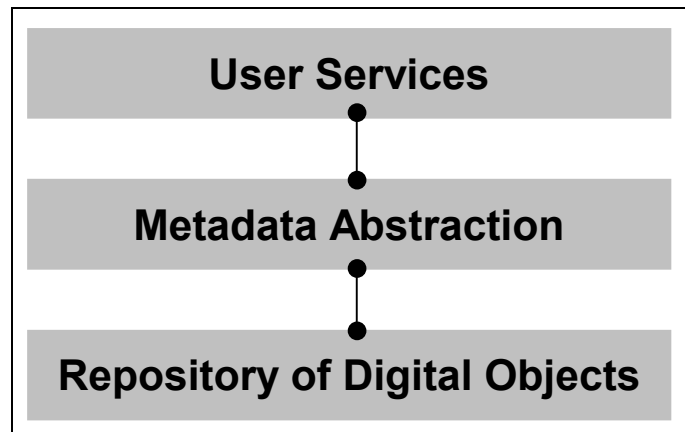


Figure 2.2 Layered organization of data storage and service provision

2.3 TECHNICAL FRAMEWORK

2.3.1 Underlying Technology and Standards

2.3.1.1 HTTP

In creating a protocol for interoperability, it was considered prudent to build upon the existing infrastructure provided by the WWW. Thus, the OAI Protocol for Metadata Harvesting is based on HTTP (Fielding, et al., 1999), closely following the model upon which HTTP is based, and leveraging its mechanisms for redirection, error handling, and parameter passing. The Protocol for Metadata Harvesting is a request-response protocol – the client makes requests for data and the server returns corresponding responses.

2.3.1.2 XML

While all requests are encoded as HTTP GET or PUT operations, responses are in XML (Bray et al., 2000) so as to allow for structure within the response data. This is especially well suited to handling the case where a service provider requests structured metadata from a data repository. The frequently thorny issue of character encoding also has been deftly avoided by utilizing the support for such features in XML.

2.3.1.3 XSD and Namespaces

Data quality and correctness of implementations are crucial to the success of any new standard. To maintain such quality, automatic and manual testing can be performed on data providers to ensure conformance to the protocol. In both instances, this testing is largely driven by precise definitions of valid XML responses in the form of XML Schema Descriptions (XSD) (Fallside, 2000). While XSD is still a very young technology, it greatly enhances the ability to specify what constitutes a valid XML document. Service providers and conformance testing tools like the Repository Explorer (Suleman, 2001) use XSD tools to automatically validate XML responses from data providers.

XML tags may be grouped together by using a prefix for each group called a namespace. Namespaces are used to support the reuse of existing semantics and schemata, making validation a modular process. For example, some responses contain metadata fields embedded within a larger structure – in these cases, the metadata will use one namespace and the rest of the XML can belong to another namespace.

```

<testxml xmlns="space1" xsi:schemaLocation="space1 space1.xsd">

  <name>Joe Smith</name>
  <comment>testxml, name and comment are in the namespace space1</comment>

  <metadata xmlns="space2" xsi:schemaLocation="space2 space2.xsd">
    <date>2000-02-28</date>
    <description>
      metadata, data and description are in the namespace space2
    </description>
  </metadata>

</testxml>

```

Figure 2.3 Fragment of XML illustrating namespaces and schema locations

Figure 2.3 is a fragment of typical XML where namespaces are used to delineate tags from different namespaces by means of “xmlns” attributes. At the same time, the schema for each namespace is indicated with an “xsi:schemaLocation” attribute that creates a mapping from the namespace to the XSD document that can be used to validate the XML.

2.3.1.4 Dublin Core

It is compulsory that all open archives be able to generate metadata for all resources in unqualified Dublin Core (DC) (DCMI, 1997). This will ensure that service providers who do not understand any other metadata format will at least be able to glean the basic information about resources from their DC renditions. Dublin Core is almost never the best choice for metadata for any given repository, but its generality makes it suitable for interoperability in the context of the OAI and its application to various different types of repositories such as papers, theses, and multimedia documents. In addition to DC, repositories also may support other optional metadata formats that are better suited to represent the objects they contain. Thus, repositories connected with NDLTD also can support MARC or ETDMS – the electronic thesis metadata standard (Atkins, et al., 2001).

2.3.2 Sets

Sets are a special construct which allow a repository to expose its internal structure to service providers. It is not compulsory for an archive to support set constructs but it provides one more mechanism for selective harvesting. There are no predefined semantics for what constitutes a set so any use of sets must be by explicit agreement between data providers and service providers. For example, in the context of NDLTD, a national archive might have sets for each region, and subsets for each university.

2.3.3 Records

A record is the metadata bundle that is associated with a unique identifier. Usually, records correspond to simple digital objects but this is not necessary – records also can

refer to collections or sub-objects. Records are encapsulated within a special structure that includes both the metadata and a header containing special fields used to support the harvesting operation. Figure 2.3 displays a typical record.

```
<record>
  <header>
    <identifier>oai:arXiv:alg-geom/9202004</identifier>
    <timestamp>1992-02-10</timestamp>
  </header>
  <metadata>
    <oai_dc xmlns="http://purl.org/dc/elements/1.1/">
      <title>Mirror symmetry and rational curves on quintic threefolds: a guide
        for mathematicians</title>
      <creator>Morrison, David R.</creator>
      <subject>Algebraic Geometry</subject>
      <description> We give a mathematical account of a recent string theory
        calculation which predicts the number of rational curves on
        the generic quintic threefold.</description>
      <date>1992-02-10</date>
      <type>e-print</type>
      <identifier>http://arXiv.org/abs/alg-geom/9202004</identifier>
    </oai_dc>
  </metadata>
</record>
```

Figure 2.4 Sample record from the arXiv open archive

2.3.4 OAI Protocol for Metadata Harvesting

The OAI Protocol for Metadata Harvesting supports 6 service requests or “verbs” that may be submitted to a repository. The protocol specifies the formats for HTTP queries and XML responses for each of these. These service requests, along with their parameters and descriptions, are listed in Table 2.1.

Service Request	Description	Parameters
Identify	Return information about the repository, e.g., name of the repository, the protocol version, administrator's email address. There also is an extension mechanism for a repository to specify additional information by supplying its own schema.	(none)
ListMetadataFormats	List all metadata formats supported by the archive, or all the metadata formats in which a particular object may be rendered.	identifier – object for which to list metadata formats.
ListSets	List the sets (and subsets, recursively) contained within the repository.	resumptionToken – token to get next batch of sets
ListIdentifiers	List identifiers for all objects or, if specified, those within a given date range and/or within a given set.	set – the set to list identifiers from from – starting date until – ending date resumptionToken – token to get next batch of identifiers metadataPrefix – format to use (version 2.0 only)
GetRecord	Retrieve the metadata for a single object in a specified metadata format.	identifier – object for which to return metadata metadataPrefix – format to use
ListRecords	List complete metadata for all objects or, if specified, within a given date range and/or within a given set.	set – the set to list identifiers from from – starting date until – ending date resumptionToken – token to get next batch of records metadataPrefix – format to use

Table 2.1 Service requests in the OAI Protocol for Metadata Harvesting

2.3.5 Flow Control

In principle, the OAI subscribes to the philosophy that the act of a service provider harvesting a repository ought not to interfere with the regular use of the archive by users through, for example, an existing WWW-based search and retrieval interface. However, some service requests have the ability to return very long response sets, e.g., **ListRecords**. So, to prevent overloading, the data provider can break result sets into chunks and return one chunk per request with a token being passed to keep track of the state of the system. Additional tags were introduced in version 2.0 of the OAI-PMH to indicate the position of the result set fragment within the complete set of results and to specify an expiration date for the token. Other flow control mechanisms, like the ability to redirect a request or the ability to postpone a request, are inherited from the underlying HTTP protocol.

2.3.6 Registration Services

Registration of conformant repositories is useful within communities with shared interests. For example, NDLTD will have a listing of all its member institutions that implement the OAI protocol. Registration can be automated by using the Identify service request to return information about an archive. On a more global scale, the OAI is attempting to register all repositories in order to possibly provide a name resolution service from identifiers to repositories. While not a requirement for all archives, OAI-PMH v2.0 recommends also the use of fully qualified domain names for repository identifiers, thus delegating the task of maintaining uniqueness to an existing naming authority.

2.3.7 Expansion and Customization

The protocol has optional features in some strategic places to allow for future expansion. Most importantly, there is no restriction on which metadata formats may be supported as long as each one has an associated schema description. Also, the data returned by the **Identify** request includes optional sections for descriptions that conform to external schemata. Similarly, each record has an optional <about> section that may contain information about the metadata object, as opposed to the digital object associated with the metadata. Figure 2.5 displays a minimal metadata record with this optional section.

```
<record>
  <header>
    <identifier>oai:arXiv:alg-geom/9202004</identifier>
    <datestamp>1992-02-10</datestamp>
  </header>
  <metadata>
    <oai_dc xmlns="http://purl.org/dc/elements/1.1/">
      <title>Mirror symmetry and rational curves on quintic threefolds: a guide
        for mathematicians</title>
      <creator>Morrison, David R.</creator>
    </oai_dc>
  </metadata>
  <about>
    <oai_dc xmlns="http://purl.org/dc/elements/1.1/">
      <creator>University Library Cataloguing Service</creator>
    </oai_dc>
  </about>
</record>
```

Figure 2.5 Minimal metadata record from arXiv with optional <about> section

2.4 REQUIREMENTS TO BE A PROVIDER

2.4.1 Data Provider

Any archive that wishes to become a Data Provider must satisfy a few basic requirements. Firstly, and most importantly, the archive must have an online interface and a Web server that can be used for the purposes of the protocol. Then, each record in the archive must be persistent or at least must contain a persistent identifier, each of which must be unique within the archive. It also is highly recommended that each

archive have a unique archive name embedded within its identifiers for records so that OAI records can be globally unique – the OAI protocol suggests that unique identifiers adopt the form “oai:archive_id:record_id”. Finally, every record must have an associated date stamp to allow for harvesting of records within a particular date range.

2.4.2 Service Provider

Service providers may use the data they harvest as they wish to, within the boundaries laid out by the data providers. While the protocol does allow for an entire archive’s contents to be harvested, it is expected that service providers will use date ranges to incrementally harvest new additions to a repository. This is illustrated in Figure 2.6

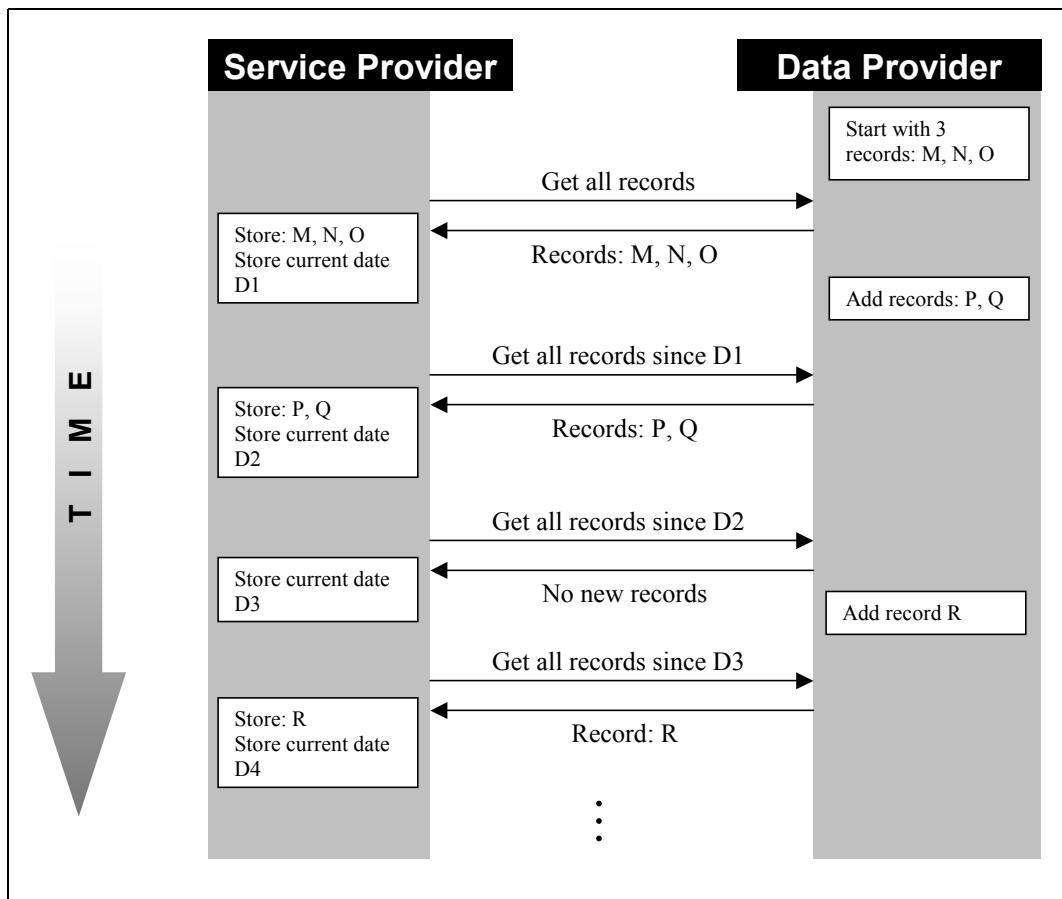


Figure 2.6 Example sequence of requests and responses between service and data providers

2.4.3 Tools and Support

The OAI website contains links to a number of useful resources that may assist developers in making their archives compliant with the protocol. The Repository Explorer is a tool that allows a user to interactively browse through an archive using only the OAI interface, while checking the interface thoroughly for errors in encoding or protocol semantics. There also is information on joining a mailing list of developers,

who are willing to share their code and expertise in various programming languages and on various platforms to ease the process of developing Open Archives. There is a growing library of tools and sample implementations to support new adopters of the technology.

2.5 OAI SUPPORT FOR TYPICAL SERVICES

2.5.1 Cross-Archive Searching

The most obvious service to provide would be cross-archive searching. The service provider can harvest metadata in one or more formats from multiple remote OAs and index the data according to collection, set, or specific fields within the metadata. Such an experimental search engine has already been developed at Old Dominion University (Liu, 2001) in parallel with the development of the OAI protocol. Other projects such as TORII (Bertocco, 2001) and OLAC (Bird and Simons, 2001) provide cross-archive searching as one of the services offered to their users.

2.5.2 Reference Linking

The ability to navigate quickly from one electronic publication to another that it references is a goal of many reference-linking techniques such as SFX, developed at the University of Ghent (Van de Sompel and Hochstenbach, 1999). OAI-accessible bibliographic metadata will greatly improve the quality and quantity of data available for constructing cross-reference databases. References can even be augmented or replaced by OAI identifiers, with an appropriate name resolution service to redirect the user to the DL that contains the referenced object.

2.5.3 Annotations

Since annotations are additions to existing documents, adding such a service to an existing DL usually requires the construction of a separate annotation database. In leveraging the OAI protocol, such a separate database can itself be an OA – then any entry in the OA of annotations will refer back to records in other existing OAs. A service provider then can retrieve data from both the source OA and annotation OA before displaying the metadata to the user.

2.5.4 Filtering

In a profile-based filtering system, users can indicate a set of interests and then all objects corresponding to those interests will be presented to them on a continuous basis. This mode of operation is perfectly suited to the OAI protocol because of the inherently incremental nature of harvesting. Thus, a filtering or routing system can use the OAI protocol to harvest new metadata and then route that as appropriate based on a set of stored profiles.

2.5.5 Browsing

Unlike searching, a browsing service often requires that the metadata contain fields with controlled vocabularies that can be used to build categories within which the objects may be placed. The support for arbitrary metadata formats in the OAI protocol allows embedding of categorical data into an appropriate metadata format. In addition, the requirement for strict conformance to an XML schema can ensure that a controlled vocabulary is adhered to.

2.6 DIGITAL LIBRARY POLICIES FROM AN OAI PERSPECTIVE

2.6.1 Ownership and dissemination control over digital objects and metadata

One of the major concerns that librarians have about this technology is its impact on ownership of digital objects and metadata. Some archives will openly share both with all and sundry while many archives will only share their metadata. There also are many archives that will share metadata for the purposes of building cross-archival search services but insist on users switching over to their website for the purpose of enforcing “brand recognition” or to request payment for resources. All of these scenarios are feasible since the OAI requires only that the metadata point to the object, and this can easily be in the form of an indirect link through the originating archive. In the case of an archive that needs to restrict access to only a specified set of service providers, that can be accomplished through the access control mechanisms built into the HTTP protocol. These allow a server to accept or reject HTTP requests based on the requesting client’s IP address or based on a name and password authentication mechanism. For additional security, the SSL protocol (Freier, et al., 1996) may be used to encrypt all transmissions.

2.6.2 Changes and withdrawal

Besides ownership, most archives also reserve the right to make changes to the metadata that is associated with their digital objects. In order to propagate changes, all an archive needs to do is update the date stamp on the record so that future requests for incremental changes will result in the changed record being disseminated once again to service providers. Service providers are expected to understand that a record received with the same identifier as a previous one is an updated version. Deletions are handled in a similar way in the OAI protocol – if identifiers for deleted records are stored at the archive, these can be returned to service providers with a special attribute that is set to indicate that the record has been deleted at the source.

2.6.3 Preservation

Preservation of digital objects is a basic requirement of the OAI. Any archive subscribing to the OAI model of interoperability must maintain a stable collection of digital objects. The HTTP protocol has a feature to redirect URLs automatically - since objects are usually referred to by URLs, this HTTP feature can be exploited to preserve the integrity of metadata. Also, if it is expected that objects will change location often

during their lifetime, they can be allocated PURLs (Shafer, et al., 1996) or Handles (Sun and Lannom, 2002) instead of regular URLs to increase the level of persistence. An essential aspect of any DL is the migration of content to newer archival technology – this is vital for interoperability efforts like the OAI since inaccessible content at a data provider will adversely affect every harvester of that data provider.

2.6.4 Uniqueness of objects and collections

The OAI does not require that every implementer of the harvesting protocol have a unique archive identifier. However, this is recommended so as to create a globally unique namespace for OAI identifiers. This will allow for the creation of services that are analogous to DNS name resolution – given an OAI identifier, the resolver with full knowledge of all OAs can direct a user to the archive that contains the resource.

Within archives each record must have a unique identifier so that any single **GetRecord** request for metadata associated with the identifier will be unambiguous.

2.7 BUILDING OAI SUB-COMMUNITIES

2.7.1 Metadata formats

Communities of archives with similar interests may benefit greatly from developing their own metadata formats or simply specifying their existing metadata formats in a form that is usable with the OAI protocol. The protocol was designed to support a much higher level of semantic interoperability than is allowed by unqualified Dublin Core, so it is expected that individual archives will choose the most appropriate format for exporting their data. For example, libraries will probably use a form of MARC encoded in XML while repositories of educational resources may instead wish to use the IMS metadata format (IMS, 1999). Thus, providers of services will be able to supply users with more information, and archives will truly be able to interoperate if they have the same underlying metadata formats.

Some representatives of pre-print archives have already begun discussion of a metadata format suited for their purposes and it is hoped that this process will be initiated within other DL communities as well.

In the context of ODL, metadata formats can be defined with specific semantics to enable the provision of high-level services; examples of these are given in Chapter Four.

2.7.2 Protocol extensions

While the protocol as specified is useful for some purposes, there is no reason why an individual community cannot enhance or change the protocol to support additional features. These can take the form of either changes or additions and could be internal, with an external interface that conforms to the base protocol. Nobody expects that this protocol is a perfect solution to every problem; rather it is a stable and tested protocol that will be used for experimentation in research and production environments, leading to further evaluation and possibly newer versions in the future as supporting standards and

techniques emerge. The encoding of a protocol version into the protocol further ensures that any future updates will not confuse service providers.

ODL exploits this ability by layering additional semantics over the basic OAI protocol.

2.7.3 Shared semantics

Along with shared metadata formats a community must share a common understanding of the semantics of each chosen metadata format. Thus, for example, if a community decides to use the RFC1807 metadata format (Lasher and Cohen, 1995), some loosely defined fields can be further restricted for the purposes of the community, thus allowing for a more tightly coupled interoperable environment. Of course, the parallel DC metadata set must still be supported so this creates the situation where an archive may export its data in a well-defined community-specific format and a loosely defined general format satisfying the general OAI community.

2.8 CASE STUDY: OAI IN THE NDLTD COMMUNITY

2.8.1 Context

NDLTD, the Networked Digital Library of Theses and Dissertations, (Fox, 1999; Fox, 2002; Fox, et al., 1996; Fox, et al., 1997; Suleman, et al., 2001) is an international alliance of universities where students submit electronic versions of their theses and dissertations. As a preliminary step towards creating a universal catalogue of publications, the community has defined a metadata standard to meet its particular needs. This metadata standard, ETDMS – the Electronic Thesis and Dissertation Metadata Set (Atkins, et al., 2001), is an extension of Dublin Core with additional fields for the provision of information about the type of thesis or dissertation. The fields inherited from Dublin Core are given specific semantics that will be understood by all members of the community. Ultimately, this metadata format will be exported from all NDLTD sites that are accessible through the OAI Protocol for Metadata Harvesting.

2.8.2 Development of OAI MARC format

In order to support libraries that are part of NDLTD, an XML version of the US-MARC metadata format has been specified in a cooperative effort between Virginia Tech's Digital Library Research Laboratory and Herbert Van de Sompel (then at Cornell University). This mapping does not attempt to encode each MARC field into a separate XML tag, but rather encodes the fields as name/value pairs, with subfields used as required. See Figure 2.7 for a fragment of oai_marc XML (Van de Sompel, 2000).

```

<oai_marc xmlns="http://www.openarchives.org/OIA/oai_marc" status="n" type="a"
level="m" catForm="a">
  <fixfield id="1">"tmp96303807"</fixfield>
  <fixfield id="3">"OCoLC"</fixfield>
  <fixfield id="5">"19970728102440.0"</fixfield>
  <fixfield id="8">"971114s1996 dcu f000 0 eng d"</fixfield>
  <varfield id="35" i1="" i2="">
    <subfield label="a">l258-02760</subfield>
  </varfield>
  <varfield id="40" i1="" i2="">
    <subfield label="d">GPO</subfield>
    <subfield label="d">DLC</subfield>
    <subfield label="d">MvI</subfield>
  </varfield>
  <varfield id="49" i1="" i2="">
    <subfield label="a">VP11</subfield>
  </varfield>
  <varfield id="74" i1="" i2="">
    <subfield label="a">0378-H-12</subfield>
  </varfield>
  .
  .
  .

```

Figure 2.7 Fragment of sample record of XML encoding of MARC

The biggest challenges were in encoding of the character sets. Since the XML style recommended by the OAI is to use Unicode entities, all ANSEL characters need to be translated into Unicode before being exported. Composite characters also need to be changed since they are encoded differently in MARC and XML. Nevertheless, this MARC encoding in XML has generated much interest from librarians because of its simplicity and the fact that any problems can be fixed at a level outside of the schema description. Further work has resulted in a new standard recently developed by the Library of Congress and included as a recommendation for version 2.0 of the OAI-PMH.

2.8.3 MetaLibraries

In a library environment, cataloguing information is a vital resource that is shared among libraries. The OAI protocol provides a low barrier method of exchanging such cataloguing information without having to invest in high-end technology solutions. The existence of the oai_marc encoding further simplifies the task since there is now a standard way of transferring MARC records in XML.

While this may not appear very useful to large research and even public libraries, it can be very useful for smaller organizations that operate libraries. It provides a means for these smaller libraries to share their metadata with larger as well as peer institutions. Conceptually, it is even possible for an appropriate organization to make available a “metalibrary” catalogue that describes every book in every OAI accessible library. This has been done as part of the ODL prototype system and is discussed in later chapters.

2.8.4 Name authority systems

The authoritativeness of names is always a problem when dealing with large quantities of data that contain references to individuals. One solution is to maintain a central (or

distributed) database of names (personal and institutional) and then use links to this in each metadata item. NDLTD has adopted this approach and is currently working with OCLC (OCLC, 2002a) to set up such a system (Suleman, et al., 2001). While name information is not usually considered to be metadata, the OAI protocol can be used for name lookups by issuing **GetRecord** requests with the name identifier as the parameter. This is being pursued actively and illustrates another scenario where the OAI protocol can be used for simple metadata access by identifier. The OAI protocol is usually restricted to batch transfers of metadata, but in this case it is used as a repository access protocol, functioning as the interface between the name authority database and users of such information.

2.8.5 Search and Classification for ETDs

NDLTD comprises a number of research universities with collections of electronic theses and dissertations. These collections are, however, managed as independent projects, very loosely linked. As an initial attempt to develop a cross-archive search service, Powell and Fox (Powell and Fox, 1998) created a federated search system. This suffers from the problem of scalability since each new archive can introduce new search semantics that will need to be integrated into the rest of the system. Also, there is no easy means of integrating the results from different systems into a single result list.

As an alternative approach, Virginia Tech is working with VTLS (VTLS, 2002) to develop a cross-archive search system based on their Virtua software. This project is using the OAI protocol to transfer metadata from individual ETD repositories into a central NDLTD collection that subsequently serves as a data source for Virtua and Virginia Tech's IR research system, MARIAN (France, 2001). In this instance, OAI technology is bridging the gaps among various different archives to increase the visibility of scholarly publications.

2.9 FUTURE DIRECTIONS

The Open Archives Initiative has provided the community of electronic libraries with a simple but extensible protocol to facilitate interoperability. But why do we need interoperability? The short answer is that there are very few digital libraries that have both extensive collections and effective services. Some contain lots of data. Others provide lots of services. In either case, users do not easily find the resources related to their particular information needs. Through OAI we can turn these problems into advantages by helping both data providers and service providers do a better job at their specialties, while streamlining the data provider to service provider connection. By building interoperable DLs, we can provide users with the best of both worlds, making searching of DLs a feasible notion without compromising on the quality of information management that sets digital libraries apart from the mass of data on the WWW. Part of this entails open exportation of data using the OAI protocol. This approach has received much support from the DL community in general. Just as important, however, is the development of high quality services that operate over the collections of data. This is progressing at a much slower pace. The Open Digital Library project seeks to design an architecture that will make the provision of services as simple as the provision of data,

thus eliminating the bottleneck in development, management and interoperability of digital libraries, ultimately enabling the participation of users all over the globe in the new “Information Age”.

Chapter 3

ODL DESIGN CONSIDERATIONS

3.1 INTRODUCTION

An Open Digital Library is a network of extended Open Archives that work together to supply the services required by information seekers. Each node within this network is an ODL service, designed to conform to a set of principles for maximum reusability at the levels of design, implementation, and information sharing.

The Open Digital Library design framework allows for building digital libraries in a systematic manner based on current principles of good software engineering, in the context of the state of the art in digital library services and networked information provision. The ODL design specifies strategies for the construction of modular DLs in a new information society where we can no longer afford to consider issues such as interoperability to be optional afterthoughts.

As previously discussed, this approach to building digital library systems is motivated by numerous factors, including the emergence and growing popularity of the Protocol for Metadata Harvesting (PMH) designed and championed by the Open Archives Initiative (OAI). The ODL is based on extensions of this protocol to provide maximum reuse of existing standards in support of greater functionality and applicability to domains not normally associated with metadata harvesting.

In an online community where new standards are being created all the time, ODL presents not just one more standard, but a set of guidelines for building on what exists – an elaborate attempt to define simpler protocols that will work rather than complex protocols that are complete. But, at the same time, ODL can be contrasted with other interoperability protocols to illustrate how these extensions enable a natural mapping to known technology.

This chapter discusses principles of ODL and its parallels to other technologies, as a prelude to a concrete mapping of DL services to ODL service components in the next chapter. It begins with a discussion of the influences of the Internet on ODL, followed by a specification of what an ODL protocol conforms to, and concludes with a mapping to a programming paradigm to demonstrate a degree of completeness.

3.2 ODL VS. THE INTERNET: A PRACTICAL PERSPECTIVE

ODL is not in itself a new idea. It is, however, the application of an idea from fundamental domains in computer science to an emerging field that is on the periphery of the discipline, namely Digital Libraries. There are stark similarities between ODL and

the way in which the Internet has evolved, leading to a set of guiding principles, which may serve as indicators of probable success.

The success of the Internet as a large-scale network of interoperable systems cannot be ignored. Of course many people believe that this success is the panacea for our problems and we need not invent new technology. The counter argument is just as compelling – there are many who believe that a little more structure would aid greatly in supporting innovative user services. As an example, the Semantic Web project (Berners-Lee, et al., 2001) is an attempt to add meaning to the links in the WWW to aid with machine understanding of the tangle of content. The OAI approach is to build layers over the Internet to add delicate shades of meaning that will support data transfer operations within the archiving community. ODL components, in turn, add meaning to the OAI protocol to support more complex and useful services than just data transfer. Figure 3.1 illustrates this encapsulation.

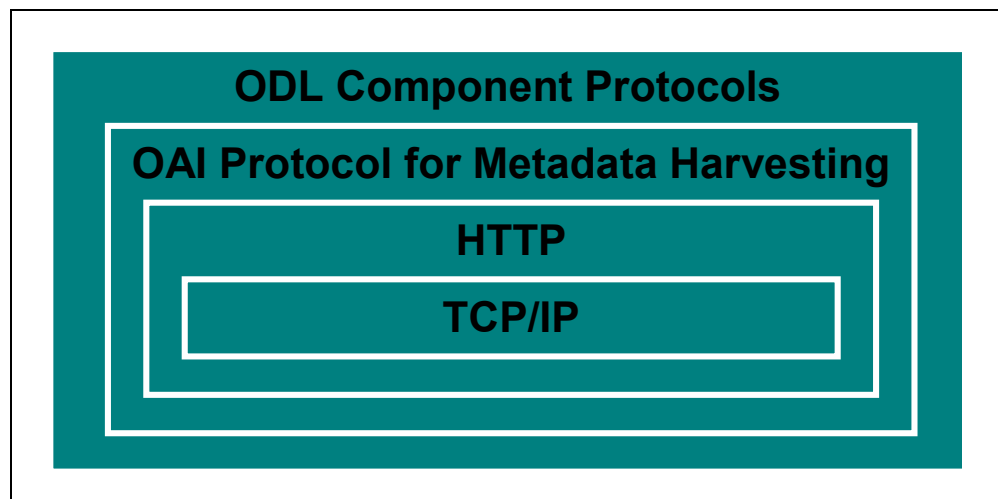


Figure 3.1 Encapsulation of network protocols, up to the level of ODL

This layered approach to building standards assumes an implicit inheritance of principles from one layer to the next. The OAI protocol adopts many of the design methodologies of the HTTP protocol. The parameters are passed using a well-established standard called the Common Gateway Interface or CGI (Gundavaram, 1996) and errors are returned via the HTTP protocol mechanism. There are many pre-defined error messages in HTTP and as far as possible these are used to indicate OAI errors. The advantage of this approach can be stated best in the language of the Dublin Core community, which recommends that it be possible for extensions to standards to be “dumbed down” to the original standard (DCMI, 2000). Thus, while it is possible to access an OAI archive using an OAI client like the Repository Explorer (Suleman, 2001), it is still possible to use an HTTP client. Similarly, in the ideal case, it ought to be possible to “dumb down” any ODL protocol into the OAI protocol (and, transitively, into the HTTP protocol).

To support this layering, ODL must be designed to be consistent with Internet and OAI protocol design. Design principles from the former can be extrapolated from the way in which individual services have evolved to effectively serve the needs of users. The

following principles are prime examples of issues in design that need to be carried through to ODL protocol design.

3.2.1 Simplicity

The prime driving force behind the Internet was its simplicity, where each component was constructed in such a way that it can be understood as a single, complete, and independent entity (Berners-Lee, 1996). The protocols are minimalist and in many instances new protocols start off with very few features and more complexity evolves only after acceptance. Many new protocols are being designed to be complete solutions and they fail not because of technical inadequacies but because of reluctance of the targeted community to accept a new complex standard. The Open Archives Initiative was launched as a direct response to this problem – its principle mission focusing quite specifically on establishing a low-barrier to interoperability (Lagoze and Van de Sompel, 2001). ODL must of necessity continue the tradition of this “low-barrier” to adequately address the needs of the community for DL components.

3.2.2 Openness

The Internet operates on the basis of a set of standards that are owned by the community itself. There is no standards body that summarily issues new standards. Instead, each standard is written as a “Request For Comments” and posted for discussion (RFC Editor, et al., 1999). On the surface, the obvious problem with this solution is one of management and quality control. However, as evidenced by its success and the success of the emerging open source community, peer acceptance and feedback are usually sufficient motivation to support the process of creation, whether of new standards or new software (Raymond and Young, 2001). The ODL design should therefore build on open standards and itself be open to comment and revision, to gain maximum benefit from the *modus operandi* of the networked community.

3.2.3 Independence of protocols

POP (Myers and Rose, 1996) and IMAP (Crispin, 1996) are two examples of very popular Internet protocols that allow a client program to access a remote mailbox. While both of these rely on the fundamental TCP/IP layer, they do not have any requirements for collocation, development, or installation in parallel – they are completely separate protocols. This has far-reaching consequences in that each protocol is allowed to develop separately of the other. As it turns out, each protocol is attributed to a separate group of people and, while they share some ideas, the different services are driven by different goals (Gray, 1995). The DL community has not generally adopted this idea. In general, most DLs tend to attempt provision of a wide range of services using the same base of code, protocols, and data storage. As a result, many compromises have to be made to support multiple collocated services. A typical example of this is the DL based on a database, which uses SQL queries for searching and browsing instead of implementing a separate IR system for searching (e.g., CSTC, WCR, E-Prints). ODL adopts the notion of independence of services from the Internet and advocates that every service be

completely separated so as to maximize the quality of that individual service without having to make compromises to support others.

3.2.4 Loose coupling

When analyzing existing DL systems, one of the greatest failings is the degree to which individual modules are coupled. This coupling of various services is motivated by the fact that a wide range of functions can be shared among modules when the system is being built from scratch. However, if the system is not being built from scratch, tightly coupled modules are not easy to reuse. To address this, modular development can be extended to the level of componentization, with loose coupling and well-defined communication protocols so that the development of systems shifts from programming to composition of independent components (Nierstrasz and Dami, 1995; Szyperski, 2000). This has already manifested itself on the Internet where all popular services communicate through well-defined protocols. Learning from those experiences as well as the experiences of the software development community, such componentization needs to be an intrinsic part of an ODL.

3.2.5 Layers

Computer networking evolved from a motley collection of network protocols to the well-understood system that is TCP/IP. With both practice and the theory of the ISO's Open Systems Interconnection model (ISO, 1994) backing it, TCP/IP has proven that layering of protocols does in fact work – that the levels of encapsulation, processing, and sometimes redundant information introduced has a payoff in terms of ultimate deployment of the technology. ODL uses a similar layered approach, with protocols being devised as extensions of the OAI-PMH or other existing ODL protocols. As seen in the somewhat offbeat Hypertext Coffee-Pot Control Protocol (Masinter, 1998), it is possible to design a protocol as a layer over HTTP to accomplish a very different goal, but without redefining the basic semantics of HTTP. ODL components apply the same philosophy to proposed extensions of the OAI protocol.

3.2.6 Reuse

Reuse is important both in the context of software and design methodologies. In terms of software reuse, the Internet has proven time and again the software engineering mantra of reusing existing software in the development of new projects. As an example of such, there are numerous projects, such as the Perl and PHP interpreters, that have adopted and incorporated the Expat XML parser (Cooper, 1999) rather than reinvent the wheel.

From the design perspective, Internet protocols may be designed separately, but the vocabulary and even semantics are retained across projects as reuse greatly diminishes the learning curve. It is counter-intuitive to introduce a new and unknown vocabulary for every component, so ODL components are designed to reuse semantics established by prior components if at all possible.

3.2.7 Orthogonality with a Purpose

From a theoretical perspective, orthogonality helps in creating complete models of a software system. But orthogonality in a protocol is not always efficient and, from a pragmatic perspective, not necessary in some cases. The HTTP protocol, for example, allows a user to obtain a document given its location but does not support finding a document's location, given its name. This is perfectly reasonable since the latter is best done by alternate means (e.g., search engines), thus enabling an optimization of the server for the functionality that is most prominently requested. ODL designs follow a similar philosophy of design to address real use cases, rather than design for the sake of completeness.

3.3 OPEN DIGITAL LIBRARY DESIGN

Based upon the experiences gleaned from the development of the Internet, better models for developing networked information services can be derived. One such model is the foundation for the Open Digital Library – a set of basic principles for the development of componentized digital libraries. The ODL design principles guide a generalization of the OAI protocol so that it may be used for purposes that go beyond its original intention, namely to provide higher-level DL services. These basic principles are as follows:

1. *All DL services must be encapsulated within components that are extensions of Open Archives.*

- This is in keeping with the desirable design goals discussed above.
 - ◆ Simplicity – The OAI protocol is simple and well understood.
 - ◆ Openness – The OAI protocol is already in the public domain.
 - ◆ Independence of protocols – By designing each DL service within its own Open Archive, they are forced to be separate and separable.
 - ◆ Loose Coupling – Open Archives are accessed at a high level through HTTP, which is stateless – thus each DL service also will be stateless and only loosely coupled with other components.
 - ◆ Layers – An Open Archive need not be a single entity – it can contain other Open Archives or work collaboratively with them. Each DL service built as a layer above other services can retain its essential semantics and extend them only when required.
 - ◆ Reuse – Building systems as components leads to reuse of the end-results as well as the intermediate results such as the design process.
 - ◆ Orthogonality – The OAI protocol requires a practical degree of orthogonality and this requirement will be inherited by any service DL that builds on the OAI protocol.

- Examples of components: Search Engine, Browse Engine, Recommendation Engine, Rating Service, Review System, Annotation Engine
2. *All access to the DL services must be through their extended OAI interfaces.*
 - Search engines can use the *set* parameter to encode a query and then return the search results as either a set of metadata records or a set of identifiers depending on whether the **ListRecords** or **ListIdentifiers** request was issued.
 - Recommendation engines can use the *set* parameter to specify the user for whom recommendations are being requested. If the *set* parameter is not provided, then system-wide recommendations can be provided.
 3. *All DL services must get access to other data sources using the OAI protocol.*
 - Search engines can be connected to data sources by providing them with the baseURLs of the OAI interfaces to the collections. The search engines will then use OAI harvesters to collect metadata in specified formats from the data providers to build their inverted files or other internal representations of the collection.
 - Recommendation engines can similarly be connected to data sources and use OAI harvesters in order to obtain metadata records from the data sources. In this case, the DL services can create pre-optimized recommendations for specific users or user classes.
 4. *The semantics of each ODL protocol must be extended or overloaded as allowed by the underlying protocol, but without contradicting the essential meaning.*
 - Search engines can use dynamic sets in order to encode queries. This is not part of the OAI protocol, but the OAI protocol does not prohibit this use.
 - Recommendation services can use the optional nature of the *set* parameter to distinguish between individual recommendations and global recommendations. Once again, this is not specifically prohibited and is in keeping with the essence of set semantics.
 5. *Digital Libraries must be constructed as networks of extended Open Archives.*
 - Each node of the network will be an extended Open Archive, while each link will be accessed through the OAI protocol (with extensions if necessary). Figure 3.2 provides an example of this architectural model.

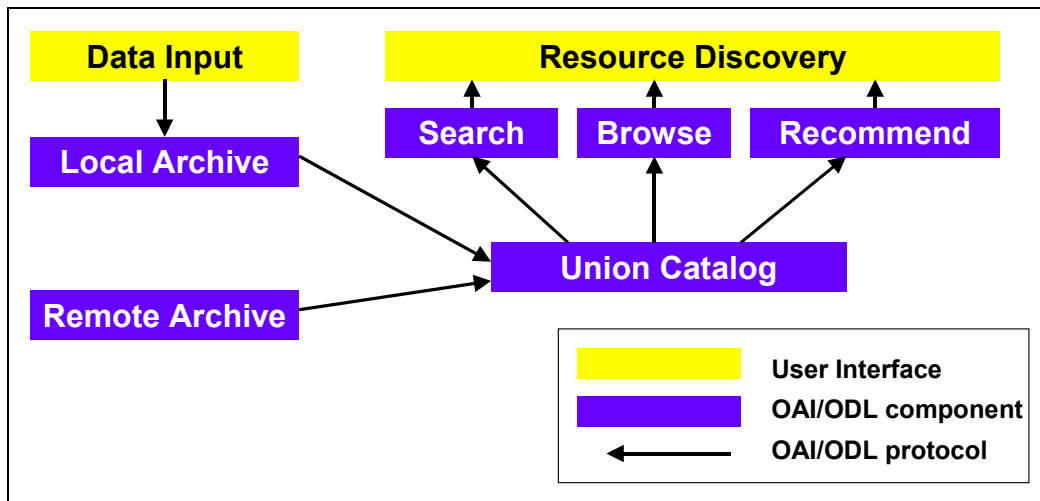


Figure 3.2 Example networked architecture of an Open Digital Library

3.4 ODL VS. OOP: A THEORETICAL PERSPECTIVE

3.4.1 Overview

From the perspective of programming language design, there are striking similarities between ODL and Object-Oriented Programming (OOP). A complete mapping between these two technologies suggests that properties of one can be restated in the vocabulary of the other, providing a theoretical backdrop for ODL. Table 3.1 shows such a mapping, with short descriptions of each concept immediately following.

ODL	OOP
Protocol	Class
Service	Instance or Object
Service Request	Message
Service Request Handler	Method
Remote Service Request	Remote Method Invocation
Service Name Resolution	Binding
Service Request Parameters	Method Parameters
Semantic Overlays	Inheritance
Dumb-Down Principle	Polymorphism

Table 3.1 Mapping of concepts from ODL \leftrightarrow OOP

3.4.2 Mappings

3.4.2.1 Protocol \leftrightarrow Class

An OOP class is the type definition that specifies how to interact with objects of that type. Similarly, an ODL protocol is the definition that specifies how to interact with services that follow the specified protocol.

3.4.2.2 Service \leftrightarrow Instance

An OOP instance is a run-time encapsulated analogue of a class, containing data and methods that operate on it. Similarly, an ODL service is the run-time encapsulation of functionality specified by a protocol, and the interface to access that functionality.

3.4.2.3 Service Request \leftrightarrow Message

An OOP message is a request passed to an object to invoke one of its methods. Similarly, an ODL service request is a request passed to a service to invoke one of its request handlers.

3.4.2.4 Service Request Handler \leftrightarrow Method

An OOP method is a procedure or function that provides a functional interface to the outside world. Similarly, an ODL service request handler provides functionality to the calling components.

3.4.2.5 Remote Service Request \leftrightarrow Remote Method Invocation

An OOP remote method invocation is when the caller of a method is remotely located. An ODL remote service request occurs similarly.

3.4.2.6 Service Name Resolution \leftrightarrow Binding

In OOP, binding is when the correct method is chosen in response to a message. In ODL components, service name resolution is when the correct service request handler is called in response to a service request.

3.4.2.7 Service Request Parameters \leftrightarrow Method Parameters

Since service requests are analogous to methods, their parameters are similarly analogous.

3.4.2.8 Semantic Overlays \leftrightarrow Inheritance

In OOP, inheritance is when an object acquires the properties of another as a precursor to building upon what already exists. An ODL component, similarly, builds upon OAI and other ODL components by overlaying additional semantics.

3.4.2.9 Dumb-Down Principle \leftrightarrow Polymorphism

In OOP, polymorphism is when an object can be used as if it were any of its predecessors. Similarly, the “dumb-down” principle implies that an ODL service may be accessed through the semantics of any of its predecessors.

3.4.2.10 Base Protocol \leftrightarrow Base Class

An OOP base class is analogous to a base protocol in ODL from which other protocols are derived.

3.4.3 Implications

With so much in common, it is also possible to explain some of the issues that affect ODL in terms of its OOP counterpart.

For example, in OOP, there is sometimes a root base class from which all other classes are derived. This is sometimes called an “Object” or “TObject” class. In ODL, this root base class is the OAI protocol, from which a whole suite of protocols can be derived in order to provide extended functionality, in a manner analogous to OOP inheritance.

As another example, OOP has the concept of pure virtual functions, which cannot be called without being overridden. In the OAI protocol, a prime example of an analogy to this is the **ListSets** service request, for which a handler has to be defined in the context of a particular archive before such service requests can be handled. Many implementers simply ignore this and use an empty response while others generate a result list by various means. Implicitly, in each of these cases, the implementers are defining a layer of additional semantics over the OAI protocol – what in OOP would be subclassing – before creating the service. In OOP this is the familiar definition of a function to override a pure virtual function before creating instances.

These typical use cases illustrate the natural manner in which the OAI protocol lends itself to a hierarchical system of layered semantics. This is the very basis for the design of ODL and such a system of layered semantics is presented in detail in the next chapter as a concrete design conforming to the requirements stated in this chapter.

Chapter 4

OPEN DIGITAL LIBRARY SERVICE PROTOCOLS

4.1 ODL SERVICES AS EXTENSIONS OF THE OAI-PMH

The primary hypothesis of this work is that DLs can be built as networks of extended Open Archives instead of as monolithic systems. Based on the design principles elaborated upon in the previous chapter, it is possible to design ODL services as mappings of use-case scenarios from traditional DLs to operations performed in the context of extended Open Archives. Each of a number of popular DL services is defined in this manner in this chapter.

The framework for each of these services is based on a general model for an extended Open Archive, referred to hereafter as the XOAI-PMH (Extended OAI-PMH). Then, for each service, a general description is given, followed by a formal description of the protocols to interface with the originating data source(s), the user interface, and other peer services.

4.2 PROTOCOL DESIGN CONSIDERATIONS

4.2.1 OAI Sets as Parameters

Many ODL component parameters that do not correspond to existing OAI parameters can be mapped into the OAI *set* parameter. This allows for a service protocol to be used by standard OAI tools while not violating the generally understood set semantics. Since sets are simply subsets of an archive, specifying a list of parameters that generate that subset is still consistent with the general notion.

4.2.2 Interface-directed Responses

The purpose of building ODL upon the foundations of OAI is largely due to the fact that OAI already defines a basic format for requests and responses. In particular, the response format allows for encapsulation of a list of arbitrary records in arbitrary metadata formats as determined by the interaction between the data and service providers. This response format need not be redefined and in many cases can be presented to the user after stylistic transformation without any preprocessing. To accomplish this, the request can be structured to provide responses that are as close as possible to the requirements of the user interface. As an example, search engines can provide parameters (such as start and stop) to extract portions of the results from a longer list so that the search components need not return every result in their OAI responses.

4.2.3 Harvesting Granularity

The OAI protocol uses a timestamp in order to support harvesting by date, but the granularity is a single day since these timestamps contain only dates and not times. After taking into account the differences in timezones, a harvester has to overlap harvesting dates by at least one day in order not to miss any new entries. This is not problematic for archives being harvested to provide cross-archive services. However, within a single archive it is a problem as individual services need to appear to be consistent to the user. As an example, if a user submits a new entry to an archive and then modifies the entry on the same day, a harvester will not detect the modifications unless it overlaps its harvesting datestamp range. The other solution is to use a finer granularity. While this does violate version 1.1 of the OAI protocol, it can be applied to all ODL protocols so it will be consistent. Version 2.0 already supports such a finer granularity for datestamps. The protocol descriptions that follow are independent of the datestamp granularity but their effectiveness and efficiency depend on selecting the correct balance between overlapping harvests and using finer granularities.

4.2.4 Response-level Containers

Another feature discussed within the OAI is the use of structured data containers, at the level of the OAI response, to return information about the act of creating the response as a whole. Currently, there exist containers within each record (in the form of the `<about>` tags) where data providers may include meta-information about each metadata record. For many services this is not sufficiently general as some information pertains to the request as a whole. As an example, a search engine can return the search result rank and number of hits in these `<about>` containers but use of the latter will not be efficient or correct since it is information about the whole response rather than each record. This information is on par with the *resumptionToken* and must be encoded in a similar manner. The following protocol descriptions assume the existence of such a response-level container.

4.2.5 Submission

The OAI protocol was conceived as an add-on layer to existing systems in order to support interoperability. As a result, the only supported data access mechanism allows a system to export its data to others. There is no mechanism to input additional data into an archive through its OAI interface. This is not a problem if harvesting is the only goal of the protocol but for more general operations such as those envisioned for ODL components, it is sometimes desirable to have a general-purpose service request to add an item to the archive. This service request can take the form of a **PutRecord** request which will be symmetric to the existing **GetRecord** request. In terms of the Kahn-Wilensky Framework (Kahn and Wilensky, 1995), adding a **PutRecord** request will make the OAI-PMH into a complete “repository access protocol”.

4.2.6 Harvesting vs. Archive Access

In the context of ODL services, most operations that are performed are accesses to data through the OAI interface, as opposed to the traditional use of the OAI interface for harvesting on a continuous basis. In most cases, harvesting is still possible, though the semantics of harvesting from non-archive components is questionable. For example, while most people can easily comprehend the concept of harvesting from an archive, harvesting from a search engine is not as familiar. For other components such as recommendation engines, where the component may incorporate multiple OAI interfaces, the semantics of harvesting and its stability are not as obvious. Thus, for ODL components, their interfaces are simply specified in terms of OAI service requests, independent of the fact that the same set of service requests also may be used for harvesting operations.

4.2.7 Dublin Core Requirement

Since most of the service-level components are only accessed through their well-defined interfaces, it becomes irrelevant to map the metadata to Dublin Core. Also, many of the internal data formats used by components are not amenable to a DC mapping.

4.2.8 Customization of Components

Every ODL component must be reusable and, as such, may have a set of configurable parameters that govern its operation. These parameters must be documented and easily modifiable as the components are used in different environments and for different purposes. The mechanism by which the components are customized is, however, beyond the scope of these definitions of the components.

4.2.9 Propagation of Information

In the OAI-PMH, a datestamp is used to track changes as well as support incremental harvesting. However, in a network of Open Archives, changes do not necessarily correspond to harvesting activities. For example, with a 2-day overlap in harvesting, some of the records received on the second day may not be new records. Since the datestamp is the only way of signaling a modification, this has to be updated to propagate records to second-level archives. Thus, each incoming record intended to be forwarded to other archives must be immediately stamped with the current date and time.

4.3 EXTENDED OAI-PMH (XOAI-PMH)

The XOAI-PMH is a general extension of the OAI-PMH to support encapsulation of various DL services within OAI-like components. This extension takes the form of a set of additional parameters and service requests for version 1.1 of the OAI-PMH. In addition, some responses and other protocol information are refined or added to as specified.

Note: Version 1.1 of the OAI-PMH was used because this work was done before version 2.0 was released.

4.3.1 Global Changes

4.3.1.1 DC Requirement

Dublin Core is not a required metadata format.

4.3.1.2 Harvesting Granularity

All timestamps used for the purposes of harvesting, including timestamp tags and from and until parameters, may use the "Complete date plus hours, minutes and seconds" variant of ISO8601. The format of this is "YYYY-MM-DDThh:mm:ssTZD", as described in section 3.2 of the OAI-PMH specification.

4.3.1.3 Identify Container

A response to **Identify** must contain information about the semantics understood by the component, in the form of a protocol name and version. This is to be encoded as follows:

```
<odl-description>
  <protocol>ODL-Union</protocol>
  <version>1.0</version>
</odl-description>
```

In the case of multi-function components, this element type can be repeated.

4.3.1.4 Response-Level Containers

All responses may have additional containers to hold information pertaining to the creation of the response as a whole. These containers appear at the very end of the response and take the form of *responseContainer* tags containing any valid XML. For example:

```
<responseContainer>
  <hits>hits</hits>
</responseContainer>
```

4.3.2 Service Request Changes

4.3.2.1 PutRecord (new service request)

Semantics:

Add, modify or delete a record from the archive. If a record exists with the same *identifier* and *metadataPrefix*, replace it with the given one. Update the timestamp on the record to reflect the current local date/time of the archive. If the *status* parameter is used to indicate deletion, delete all metadata for the record from the archive.

Parameters:**identifier**

The identifier associated with the record. This is an optional field (if status is not “deleted”) and if not specified, the archive must assign a new and unique identifier for the record.

sets

A comma-separated list of sets that the record is to become a part of. This is an optional field and if not specified, the record does not belong to any sets.

metadataPrefix

The metadata prefix associated with the metadata format of the record.

metadata

The metadata record as an XML fragment. Complete schema and namespace information must be provided but the XML header must be omitted. If the metadata field is not trivially small, the HTTP POST operation must be used instead of HTTP GET to avoid limits on URL-encoded query lengths that are common in Web servers.

status

This is an optional parameter – if it is provided and its value is “deleted” then all parameters but identifier are ignored and all metadata records corresponding to this record are deleted.

Return Values:

The response contains the identifier assigned to the record just stored.

Exceptions:

If any required parameters are missing or in an invalid format, then an HTTP status-code of 400 is returned to indicate illegal parameters.

Example Request:

```
Verb=PutRecord&metadataPrefix=oai_dc&identifier=oai:ABC:123
&metadata=<test%20xmlns%3D"testns"%20xsi%3AschemaLoca
tion%3D"testschema"><title>aTitle<%2Ftitle><%2Ftest>
```

Example Response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<PutRecord xmlns="http://www.odl.org/ODL/1.1/ODL PutRecord"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.odl.org/ODL/1.1/ODL PutRecor
d http://www.odl.org/ODL/1.1/ODL PutRecord.xsd">
<responseDate>2001-10-27T19:20:30-05:00</responseDate>
<requestURL>http://an.oa.org/OAIscrip?verb=PutRecord&metad
ataPrefix=oai_dc&identifier=oai:ABC:123&metadata=&lt;test%2
0xmlns%3D"testns"%20xsi%3AschemaLocation%3D"testschema"&gt;
&lt;title&gt;aTitle&lt;%2Ftitle&gt;&lt;%2Ftest&gt;</request
URL>
    <identifier>oai:XYZ:123</identifier>
</PutRecord>
```

4.4 PREFACE TO ODL PROTOCOL DESCRIPTIONS

Each of the ODL service components will be described with the following sections:

Description: A short description of the background and functionality of the component.

Interface Protocol: The protocol used to communicate with the component.

Interoperability Issues: Issues to address regarding interoperability with peer components.

4.5 THE ODL-SUBMIT PROTOCOL V1.0

4.5.1 Description

An Open Archive is a repository that allows read access but not write access. ODL-Submit is a trivial variant of XOAI-PMH to add support for write-access to an Open Archive, allowing for the addition, modification, and deletion of items in an archive.

4.5.2 Interface Protocol

4.5.2.1 Identify, ListMetadataFormats, ListSets, GetRecord, ListIdentifiers, ListRecords, PutRecord

Inherited from XOAI-PMH / OAI-PMH.

4.5.3 Interoperability Issues

Since the read-only aspect of such components make them appear like standard Open Archives, hierarchical harvesting may be performed to aggregate data as necessary.

4.6 THE ODL-RECENT PROTOCOL V1.0

4.6.1 Description

In order to pique their interests, many DLs present users with a listing of some sample items that have just been added. This can be accomplished using a simple extension that returns records in their original format but only returns recent ones. In terms of implementation, this entails regularly harvesting from the source archive and storing only the recent entries and/or their identifiers locally.

4.6.2 Interface Protocol

4.6.2.1 Identify, ListMetadataFormats, GetRecord

Inherited from XOAI-PMH / OAI-PMH.

4.6.2.2 PutRecord

Not supported.

4.6.2.3 ListSets

ODL-Recent Results:

Empty list.

4.6.2.4 ListIdentifiers, ListRecords

Semantics:

Return a fixed, small number of identifiers or records as specified in the configuration of the component, corresponding to a random sample from among the newest records encountered by the component.

4.6.3 Interoperability Issues

Peer components conforming to ODL-Recent can be aggregated at a higher level if the higher level component harvests from each of the lower-level ones. Thus, a central DL system can use an ODL-Recent-compliant component that harvests records from each of its contributing DLs' peer ODL-Recent-compliant components.

4.7 THE ODL-UNION PROTOCOL V1.0

4.7.1 Description

In some instances metadata from multiple archives can be coalesced into a single archive to support specific requirements such as a reduction in network traffic where multiple services require the use of the metadata from remote locations. Also, where the archives are established to parallel an organization that is itself hierarchical, it may be useful to gather all the metadata into root nodes to provide centralized services across the entire organization (or subsets of it). Figure 4.1 illustrates a simple ODL network using a component that conforms to the ODL-Union protocol.

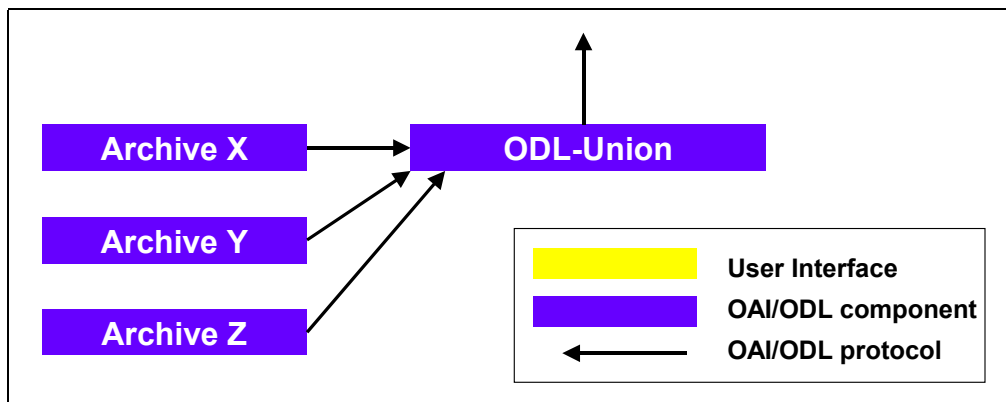


Figure 4.1 Simple ODL network using ODL-Union-compliant component

The ODL-Union-compliant component collects metadata from multiple sources and republishes it via a single XOAI-PMH interface. The algorithm used to harvest the data is implementation-dependent since the choice of algorithm affects network usage but not the logic of the protocol. Parameters may be set to select the metadata formats and sets to harvest from each source archive. From the perspective of the component, the whole archive can be harvested, thus losing set information, or the individual sets can be harvested with a more sophisticated algorithm – the component will republish whatever information it can obtain.

4.7.2 Interface Protocol

4.7.2.1 Identify, GetRecord, ListIdentifiers, ListRecords

Inherited from XOAI-PMH / OAI-PMH.

4.7.2.2 PutRecord

Not supported.

4.7.2.3 ListMetadataFormats

ODL-Union Results:

List of metadata formats representative of all records currently in archive.

4.7.2.4 ListSets

ODL-Union Results:

List of pairs of archive identifiers and source sets, where each pair is separated by a slash. For any archive harvested only as a whole, the source set will be omitted, resulting in just archive identifiers.

XOAI Response Encoding:

```
...  
<set>  
  <setSpec>archive1</setSpec>  
  <setName>archive1</setName>  
</set>  
<set>  
  <setSpec>archive2/set</setSpec>  
  <setName>archive2/set</setName>  
</set>  
...
```

4.7.3 Interoperability Issues

Components supporting ODL-Union provide almost transparent access to the data contained in each source archive. However, in order to differentiate among the various source archives, archive names are added as sets or prefixed to each existing set within an archive. As a result, any peer ODL-Union-compliant components will be linked together in a parent-child manner rather than as peers (as shown in Figure 4.2). This preserves the integrity of each source archive as a separate entity within the union archive, but it requires that if two distinct union archives acquire the same identifier/record then a higher-level union will include the disputed identifier/record in both sets.

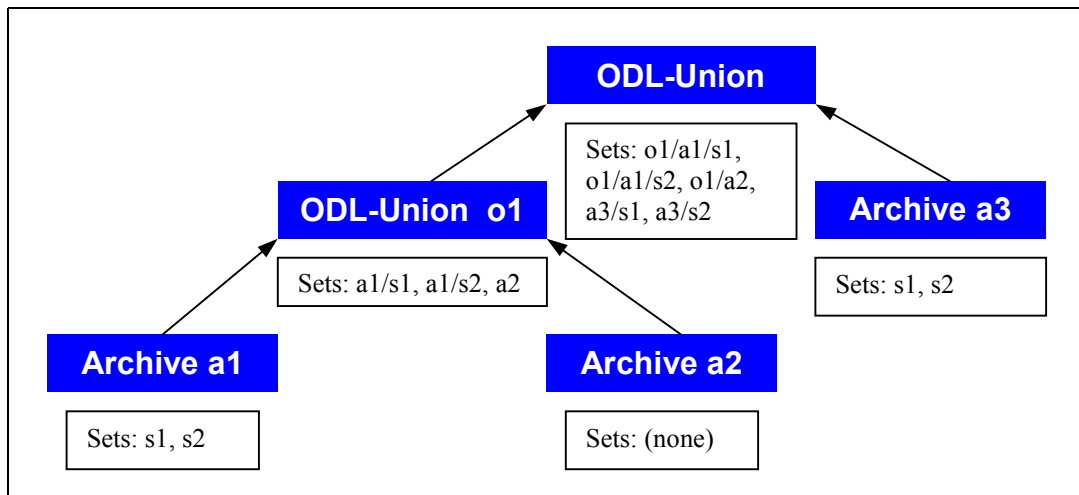


Figure 4.2 Hierarchical organization of ODL-Union-compliant components

4.8 THE ODL-SEARCH PROTOCOL V1.0

4.8.1 Description

Search engines have become a ubiquitous part of digital information seeking, reminiscent of the omnipresent card catalog in the libraries of old. Very few traditional libraries will not use an OPAC (Online Public Access Catalog) these days – and very few OPACs will not provide a function to search based on keyword or author. Nevertheless, they are still not well defined because of the wide array of functions that are attributed to search engines. In their book on search engines, Berry and Browne provide only a hint of a definition when they state that we look to search engines for “concise, organized responses” to “vague questions about topics that we’re unfamiliar with ourselves” (Berry and Brown, 1999). Kowalski, writing about information retrieval (IR) systems, enumerates the two most common views of search engines: the “fuzzy” IR system and the “structured” database (Kowalski, 1997). Brewer takes a philosophical approach and calls search engines the “fundamental solution to anarchy and chaos” (Brewer, 2001)

Given this uncertainty about what a search engine is, it is hardly surprising that DLs tend to vary widely in their interpretation of search functionality. The Greenstone system (Witten, et al., 2000) is based on the MG Information Retrieval system (Witten, et al., 1999) and so provides users with the ability to specify IR-like queries. On the other hand, the E-Prints system (OpCit, 2002) uses SQL queries so it only allows precise queries. Designing a general protocol for search engines has to take into account this variance, so the exact nature of the query must be abstracted from the protocol that supports it. The results from the query, on the other hand, may be in approximately the same format. There have been various attempts to specify interfaces for searching such as Z39.50 (ANSI/NISO, 1995) and STARTS (Green, et al., 2001), but they concentrated on completely specifying queries whereas ODL-Search concentrates on encapsulating the query process in OAI-like technology. SDLIP (Paepcke, et al., 2000) takes a similar

approach to ODL-Search, but it is defined as a standalone remote search protocol whereas ODL-Search is part of a larger suite of protocols for DL componentization.

Figure 4.3 shows a simple ODL network using an ODL-Search-compliant component.

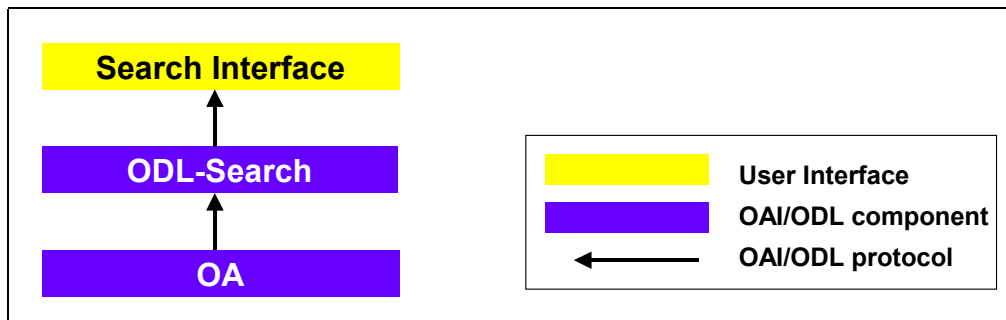


Figure 4.3 Simple ODL network using an ODL-Search-compliant component

4.8.2 Interface Protocol

4.8.2.1 Identify, ListMetadataFormats

Inherited from XOAI-PMH / OAI-PMH.

4.8.2.2 GetRecord, PutRecord

Not supported.

4.8.2.3 ListSets

ODL-Search Results:

Empty list.

4.8.2.4 ListIdentifiers, ListRecords

ODL-Search Parameters:

qlang

Name of query language used to indicate the semantics for the query that follows. The query language is a controlled vocabulary, with examples of syntax and semantics in a later section.

query

Search query in language understood by search engine.

start

Index of first item to return from complete list of results, ranked in order of decreasing relevance of the identifier/record to the query. This, along with the next parameter, allows selection of a range of results from within the complete list.

stop
Index of last item to return from complete ranked list of results.

XOAI Parameter Encoding:

set
qlang/query/start/stop

XOAI Request Encoding:

```
...Verb=ListIdentifiers&set=qlang/query/start/stop...  
...Verb=ListRecords&set=qlang/query/start/stop...
```

Additional ODL-Search Results:

hits
Estimated total number of hits. This could be the actual number, but formally defined as an estimate, it allows for approximate searching.

XOAI Response Encoding:

```
...  
<responseContainer>  
  <hits>hits</hits>  
</responseContainer>  
...
```

4.8.3 Interoperability Issues

Search engine interoperability is usually equated with the concept of federated or meta-searching – where queries are sent to remote sites, from which results are gathered and merged. In the NDLTD project, Powell and Fox built a system to support federated searching among collections of theses and dissertations (Powell and Fox, 1998). Disparities in search interfaces present one major obstacle, which can be avoided to some degree by using the ODL-Search protocol. ODL-Search thus can form the basis of a federated system of search engines. Differences in query languages will not be solved but mapping from one keyword-based query syntax to another is possible – thus existing search engines can be retrofitted with an ODL-Search interface.

4.8.4 Query Language: odlsearch1

4.8.4.1 Syntax

('+' | '-') ? (field ':') ? term (space ('+' | '-') ? (field ':') ? term) *

4.8.4.2 Parameters

field

The name of a tag in the original XML data. In the case of Dublin Core records, this can be “title”, “creator”, or any of the other 13 tags.

term

The single word on which to perform a search operation.

space

The space character.

4.8.4.3 Description

The results of a search correspond to those documents that contain one or more of the query terms, ranked in a consistent manner based on a model of relevance specific to the search engine. Query terms that are prefixed with a *field* designator must be searched for in only the corresponding XML nodes and their children. If a query term is prefixed with “+” then all results must contain that term, and if it is prefixed with “-” then no results may contain that term.

4.8.4.4 Examples

- coffee
- coffee java
- +coffee java
- coffee +java
- +coffee +java
- +computer -science
- +title:www
- +contributor:fox
- +language:ger +description:computer

4.8.5 Query Language: odlsearch2

4.8.5.1 Syntax

oai_identifier (',' oai_identifier)*

4.8.5.2 Parameters

oai_identifier

Any identifier that refers to a record in the source archive of the search engine.

4.8.5.3 Description

This mimics a feedback mechanism where a user may request more documents like a given one or a list of documents. The list of documents to use as a search query is specified in terms of the OAI identifiers. The method of extracting the query terms and merging together terms from different documents is implementation dependent.

4.8.5.4 Examples

- oai:archive1:123
- oai:archive1:a1,oai:archive2:a2,oai:archive3:a3

4.9 THE ODL-BROWSE PROTOCOL V1.0

4.9.1 Description

The definition for browsing is a contentious issue in the arena of networked information. While it is apparent from its definition that browsing entails seeking information by scanning through items, what is not apparent is its relationship to searching. Since most search engines execute queries and present the user with a list of options, browsing through those options is considered a natural post-search selection activity. However, browsing also may allow a user to scan through the entire contents of a system, a function not usually associated with searching, which tends to narrow and reorganize the search space.

With this philosophy in mind, and since the implementation requirements for browsing are vastly different from those for searching, ODL-Browse is treated as a separate protocol, with different component instantiations. While ODL-Search-compliant components can be built around information retrieval systems, ODL-Browse-compliant components can be built around indexed relational databases.

4.9.2 Interface Protocol

4.9.2.1 Identify, ListMetadataFormats

Inherited from XOAI-PMH / OAI-PMH.

4.9.2.2 GetRecord, PutRecord

Not supported.

4.9.2.3 ListSets

ODL-Browse Results:

browsable_category_1...browsable_category_n

List of browsable categories in a form suitable for submission back to the browse engine.

XOAI Response Encoding:

```
...  
<set>  
  <setName>browsable_category_i</setName>  
  <setSpec>browsable_category_i</setSpec>  
</set>  
<set>  
  <setName>browsable_category_i+1</setName>  
  <setSpec>browsable_category_i+1</setSpec>  
</set>  
...
```

4.9.2.4 ListIdentifiers, ListRecords

ODL-Browse Parameters:

- qlang
Name of browse query language used, to indicate the semantics for the browse request that follows. The browse query language is a controlled vocabulary, with the currently only defined name being “odlbrowse1”.
- criteria
Selection criteria in language understood by browse engine.
- start
Index of first item to return from complete list of results. This, along with the next parameter, allows selection of a range of results from within the complete list.
- stop
Index of last item to return from complete list of results.

XOAI Parameter Encoding:

set
qlang/criteria/start/stop

XOAI Request Encoding:

```
...Verb=ListIdentifiers&set=qlang/criteria/start/stop...  
...Verb=ListRecords&set=qlang/criteria/start/stop...
```

Additional ODL-Browse Results:

- hits
Estimated total number of hits.

XOAI Response Encoding:

```
...  
<responseContainer>  
  <hits>hits</hits>  
</responseContainer>  
...
```

4.9.3 Interoperability Issues

If two or more browse engines are working together, they need to merge the results. Since the order and meaning of the result list depends on the exact nature of the browser, this is largely dependent on implementation.

4.9.4 Query Language: odlbrowse1

4.9.4.1 Syntax

$((\text{category } (' \text{ value } (',' \text{ value }) * ')) | (\text{'sort'} (' ('+' | '-') \text{category } (',' ('+' | '-') \text{category }) * '))) +$

4.9.4.2 Parameters

category(value)

List of categories and their associated values for those categories with controlled vocabularies. This is typically used for such categories as "year".

sort(category)

List of categories without controlled vocabularies, which may form the basis for sorting but not for selection. The prefixes "+" and "-" denote ascending and descending order respectively for the category that immediately follows.

4.9.4.3 Description

The results are those documents that contain all the category/value pairs, sorted in the order specified by the sort parameter. The sort order is a list of categories on which to sort, in order from left to right, each optionally prefixed by a "+" or "-" to indicate ascending or descending order respectively.

4.9.4.4 Examples

- year(2001)
- author(Hussein Suleman)
- sort(year)
- sort(+year,-author)
- year(1997)author(Hussein Suleman)

- author(Hussein Suleman)year(2001)sort(title)
- author(Hussein Suleman)institution(Virginia Tech)sort(-year,+title)

4.10 THE ODL-RECOMMEND PROTOCOL V1.0

4.10.1 Description

The first attempts at providing recommendations were in the form of collaborative filtering where users can annotate a data source to aid other users in determining relevance of objects (Goldberg, 1992). This has since broadened into the current notion of recommender systems which are responsible for making suggestions for relevant resources when a user does not have enough expertise or experience to make those choices (Resnick, 1997).

In general, a recommender system must accept the identity of a user or subject and produce a set of potentially relevant recommendations. In the context of OAI these recommendations may be either lists of identifiers or metadata records.

The input into a recommender component depends on its internal functionality. Two possible methods of generating recommendations are:

1. to use static or dynamic user profiles based on user input or past activity, or
2. to exploit the similarities among people based on common history as a predictor of future behavior.

Each of these has different requirements for data management, hence for input into the component.

In the first case, when all that exists is a profile of preferences or past behavior, this profile has to be applied to a search engine in order to generate a set of recommendations. This mode of generating recommendations does not involve collaboration among users.

In the second case, where recommendations are generated from the similarities in behavior among subjects, there is no need to perform an explicit search operation. Instead, interactions between users and resources must be tracked and used as an indication of interest. Algorithms specific to the component implementation then can interrogate this list of interests to recommend users, based on resources and other users, and resources, based on resources and other users.

This list of records, after ranking and deduping, is considered to be the recommendation. Metadata may be retrieved if necessary from the metadata archive.

4.10.2 Interface Protocol

4.10.2.1 Identify, ListMetadataFormats

Inherited from XOAI-PMH / OAI-PMH.

4.10.2.2 GetRecord

Not supported.

4.10.2.3 ListSets

ODL-Recommend Results:

Empty list.

4.10.2.4 PutRecord

Semantics:

Informs the recommender engine that a user has expressed interest in a resource, normally through the act of viewing that resource.

ODL-Recommend Parameters:

user_identifier

The OAI identifier of the user who accessed the resource.

resource_identifier

The OAI identifier of the resource that was accessed.

XOAI Parameter Encoding:

metadataPrefix

oai_dc

metadata

<dc>

<identifier>user_identifier</identifier>

<relation>resource_identifier</relation>

</dc>

XOAI Request Encoding:

```
...Verb=PutRecord&metadataPrefix=oai_dc&metadata=<dc><identifier>user_identifier<%2Fidentifier><relation>resource_identifier<%2Frelation><%2Fdc>
```

4.10.2.5 ListIdentifiers

ODL-Recommend Parameters:

type

Type of recommendation sought: RR = resource to resources, RP = resource to users, PP = user to users, PR = user to resources.

subject

OAI identifier of the user or resource used as the basis for recommendations.

start

Index of first item to return from complete ranked list of results. This, along with the next parameter, allows selection of a range of results from within the complete list.

stop

Index of last item to return from complete ranked list of results

XOAI Parameter Encoding:

set

type/start/stop/subject

XOAI Request Encoding:

```
...Verb=ListIdentifiers&set=type/start/stop/subject...
```

Additional ODL-Recommend Results:

hits

Estimated total number of hits. This can be the actual number, but formally defined as an estimate it allows for approximate searching.

XOAI Response Encoding:

```
...  
<responseContainer>  
  <hits>hits</hits>  
</responseContainer>  
...
```

4.10.2.6 ListRecords

Semantics:

Lists all records added to the archive using **PutRecord** in oai_dc format, using conventional parameter semantics.

4.10.3 Interoperability Issues

Since all recommendation records may be harvested using **ListRecords**, it is possible to use a harvester to obtain usage information from one component and enter it into another. This can be useful for DLs to incorporate usage statistics from external sites that host copies of their metadata.

4.11 THE ODL-RATE PROTOCOL V1.0

4.11.1 Description

Many existing DL systems allow users to assign numerical ratings to an item, the average of which is subsequently displayed to other users as a simple peer review mechanism. These tend to be popular because users easily understand them. Many booksellers (e.g., Amazon) and movie websites (e.g., Internet Movie Database) have adopted the use of ratings as a simpler alternative to full-blown reviews.

ODL-Rate is used to specify ratings and generate averages for display.

4.11.2 Interface Protocol

4.11.2.1 Identify, ListMetadataFormats

Inherited from XOAI-PMH / OAI-PMH.

4.11.2.2 ListMetadataFormats

ODL-Rate Results:

odl_rating

4.11.2.3 ListSets

Not supported.

4.11.2.4 ListIdentifiers, ListRecords

Semantics:

Lists all records added to the archive using **PutRecord** in odl_rating format, using conventional parameter semantics.

4.11.2.5 PutRecord

Semantics:

Whenever a user rates an object, a record is created and added to the archive.

ODL-Rate Parameters:

subject_identifier

The OAI identifier of the user who created the rating. If this user had rated the object previously, the old rating is replaced.

object_identifier

The OAI identifier of the resource that is being rated.

num_rating
The numerical rating value assigned.

XOAI Parameter Encoding:

metadataPrefix
odl_rating

metadata
<odl_rating>
<subject>subject_identifier</subject>
<object>object_identifier</object>
<rating>num_rating</rating>
</odl_rating>

XOAI Request Encoding:

```
...Verb=PutRecord&metadataPrefix=odl_rating&metadata=<odl_rating><subject>subject_identifier<%2Fsubject><object>object_identifier<%2Fobject><rating>num_rating<%2Frating><%2Fodl_rating>
```

4.11.2.6 GetRecord

ODL-Rate Parameters:

object_identifier
An identifier for which to return the average rating.

XOAI Parameter Encoding:

identifier
object_identifier

metadataPrefix
oai_rating_average

XOAI Request Encoding:

```
...Verb=GetRecord&identifier=object_identifier&metadataPrefix=oai_rating_average
```

ODL-Rate Results:

ave
The average rating value.

num
The number of ratings on which the average is based.

XOAI Response Encoding:

```
...
<record>
  <header>
    <identifier>object_identifier</identifier>
    <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
  </header>
  <metadata>
    <oai_rating_average xmlns="..." xsi:schemaLocation="...">
      <average>ave</average>
      <number>num</number>
    </oai_rating>
  </metadata>
</record>
...
```

4.11.3 Interoperability Issues

Any ODL-Rate-compliant component can interoperate with other peer components by harvesting their `odl_rating` records and incorporating them into its database. This will work as long as the identifier schemes are globally unique or disjoint.

4.12 THE ODL-ANNOTATE PROTOCOL V1.0

4.12.1 Description

Annotations take the form of any additional pieces of metadata attached to a digital object. These annotations can take the form of reviews, ratings, corrections, or simply user comments. Some of these tasks can, however, be addressed more effectively using specialized components so this component will not address the specificity required in order to do peer-reviews.

If annotations are always attached to a digital object there is an implicit parent element for every annotation. Also, since a comment can be a reply or follow-up to another comment, it is possible to build a tree structure, mimicking the operation of the popular threaded discussion boards and newsgroups. In doing so, each annotation is considered to be equivalent to a first-class object, thus supporting a recursive operation.

4.12.2 Interface Protocol

4.12.2.1 Identify

Inherited from XOAI-PMH / OAI-PMH.

4.12.2.2 PutRecord

Semantics:

Every new annotation is submitted to the ODL-Annotate component through this service request in the form of a metadata record.

ODL-Annotate Parameters:

parent_identifier
OAI identifier of the node which is being annotated.

annotation
XML fragment corresponding to an annotation.

XOAI Parameter Encoding:

sets
parent_identifier

metadata
annotation

XOAI Request Encoding:

```
...Verb=PutRecord&set=parent_identifier&metadataPrefix=odl_annotate&metadata=<annotation><subject>subject_identifier<%2Fsubject><comment>annotation<%2Fcomment><%2Fannotation>
```

4.12.2.3 ListMetadataFormats

ODL-Annotate Results:

List of all metadata formats in the annotation archive.

4.12.2.4 ListSets

ODL-Annotate Results:

List of all non-leaf identifiers.

4.12.2.5 GetRecord

ODL-Annotate Parameters:

parent_identifier
Identifier of a node whose parent record is requested.

identifier
Identifier of single item to return.

XOAI Parameter Encoding:

identifier
parent/parent_identifier
or
identifier

4.12.2.6 ListIdentifiers / ListRecords

ODL-Annotate Parameters:

parent_identifier
OAI identifier of the parent node from which to start listing annotations in a depth-first search manner.

start
Index of first item to return from complete list. This, along with the next parameter, allows selection of a range of results from within the complete list.

stop
Index of last item to return from complete list of annotations.

XOAI Parameter Encoding:

set
start/stop/parent_identifier

XOAI Request Encoding:

```
...Verb=ListIdentifiers&set=start/stop/parent_identifier...  
...Verb=ListRecords&metadataPrefix=odl_annotate&set=start/sto  
p/parent_identifier...
```

Additional ODL-Annotate Results:

hits
Estimated total number of hits.

level
Indentation level for each record in ListRecords responses (encoded in about containers).

XOAI Response Encoding:

```
...  
<responseContainer>  
  <hits>hits</hits>  
</responseContainer>  
...
```

4.12.3 Interoperability Issues

An ODL-Annotate-compliant component can harvest annotations from any other peer component and integrate them into its collection. Since annotations can only exist for older records, the harvester will have a guarantee that it will obtain complete sub-trees during the harvesting operation. Also, harvesting has to be on a set-by-set basis since version 1.1 of the OAI-PMH does not include set information with the records. In future versions of XOAI-PMH, this may not be necessary since set membership information is contained in record headers in OAI-PMH version 2.0.

In scenarios with multiple ODL-Annotate-compliant components, whether or not a component republishes harvested annotations determines to what degree the data is filtered through the network. To get maximum dissemination, each component must publish all its harvested records along with its original records, and then when harvesting it can ignore records with their own identifiers.

4.12.4 Annotation Metadata Sub-Format: odlannotate1

4.12.4.1 Syntax

```
<text>text_string</text>
```

4.12.4.2 Parameters

text_string
Any piece of text.

4.12.4.3 Description

This language for annotations simply defines an annotation as being a piece of human-readable text. There is no location information and there are no controlled vocabularies.

4.12.4.4 Examples

- <text>This resource is really bad !</text>
- <text>The author should be barred from using a computer in the future!</text>

4.12.5 Annotation Metadata Sub-Format: discuss

4.12.5.1 Syntax

```
<discuss>
  <from>Hussein</from>
  <userid>oaiuser:1</userid>
  <datestamp>12 August 2002</datestamp>
  <subject>just testing</subject>
  <body>testing 1,2,3</testing>
</discuss>
```

4.12.5.2 Description

This is a metadata record that may be used for a discussion board or threaded forum or for simple annotations to resources in a digital library.

4.13 THE ODL-REVIEW PROTOCOL V1.0

4.13.1 Description

Peer or user reviews are a special case of annotations with very specific semantics. There is usually just a single level of entries (no threading) and the metadata stored is specific to the project and non-trivial. The ODL-Annotate component can be specialized for this purpose by defining a set of metadata formats for reviews.

However, when using a hierarchical organization for the reviewable resources, an indexed database approach is more efficient to store the non-linear data associated with peer-review workflow. The state of the system can be stored in an XML database, XML file, or a set of tables in a relational database. Then, transactions can be posted when workflow information is being updated, e.g., an editor is assigned to a resource. Reports generated from the system can be mapped to **ListRecords** requests with specific metadata formats.

4.13.1.1 Transactions

Based primarily on an analysis of the CSTC system (CSTC, 2002), but also taking into account the typical peer review process for journals and conferences, the following system has been devised to leverage the capability for multiple metadata formats in the OAI protocol to drive a reviewing system's workflow. Firstly, the set of metadata formats corresponding to the following different transactions that occur during the lifecycle of the review process are defined:

- Assign editor to resource.
- Decline to be an editor.
- Revoke editorial privileges for an editor from a resource.
- Submit a resource for review.

- Add a subsection to the hierarchy of sections and resources.
- Move a resource to a different subsection.
- Change the status of a resource/section.
- Submit a review for a resource.

Each transaction is assigned an identifier upon submission to the reviewing engine, to be used to refer to the transaction in the future. These are necessary, for example, where a user declines to be an editor but accepts a reviewing responsibility; the reviewing engine has to be able to distinguish between the two different assignments, albeit that they are for the same resource and involve the same person.

4.13.1.2 Resources and Sections

All resources reside in sections within the reviewing engine (this does not correspond to any organization of the resources external to the system) to enable the pre-assignment of editors to sets of resources. A section is a container or set that may contain resources or other sections. This allows the construction of hierarchical sections. For example, a typical section structure for a journal can be:

```
JERIC
  Volume 1
    Issue 1
      Resource a
      Resource b
      Resource c
    Issue 2
      Resource d
      Resource e
      Resource f
```

This concept can be generalized for other purposes. For a conference, the sections might be only one level deep and correspond to the various tracks of submission. An optional higher-level structure can support reviewing for multiple conferences within a series.

Each section and resource has a status flag. For sections, these can indicate if the section is open for further submissions (open), only open for reviewing of resources already submitted (underreview), or closed to both submission and reviewing (closed). For resources, the status flag indicates if the resource is under review (underreview), accepted (accepted), rejected (rejected), or awaiting resubmission from the original contributor (resubmit). New sections are automatically “open” while new resources are “underreview”. These status flags can be changed by submitting an appropriate transaction.

Metadata for resources may be stored in a different component. This protocol defines only the workflow management procedures. Thus, when a resource is submitted, the metadata is first stored in a different component and then the reviewing engine is notified by means of a corresponding transaction. If the resource’s status is “resubmit” and a new

version is submitted, the new version becomes authoritative, and the old version is stored as a draft.

4.13.1.3 Users and Permissions

Users are assumed to have their information stored in some external archive, possibly the same that is used for logins and authentication. It is assumed that every user has a unique identifier associated with him or her in this system.

A user of the system has, by default, no privileges except to submit resources to any of the currently open sections. There are two built-in boundary roles and any number of intermediate roles. The “Administrator” is the person who logs into the system first and therefore has full control over the reviewing system. A “Reviewer” is a person who reviews an actual resource. In between these two categories could be any number of editors.

An editor is a person who has reviewing responsibility for a section or a resource. Editors may assign other editors or reviewers to sections or resources as specified by an assignment control matrix (an example of such a matrix is shown in Table 4.1).

From \ To	Administrator	Managing Editor	Assistant Editor	Guest Editor	Resource Editor	Reviewer
Administrator	Y	Y	Y	Y	Y	Y
Managing Editor	N	N	Y	Y	Y	Y
Assistant Editor	N	N	N	Y	Y	Y
Guest Editor	N	N	N	N	Y	Y
Resource Editor	N	N	N	N	N	Y
Reviewer	N	N	N	N	N	N

Table 4.1 Assignment control matrix

When the system is initialized, the administrator is assigned control of a top-level node in the hierarchy. Then, the administrator may create a section, say JERIC, and assign managing editors to it. The managing editors, in turn, then create issues (and volumes) and assign guest editors to them. If there are any assistants, they can help with the process in a very specific capacity. Resource editors may optionally be assigned as specialists who can assist with individual resources by finding reviewers and making recommendations for those resources.

If a section is created (or resource is submitted) and no editor is assigned then the editors for the immediately containing section automatically have authority over the new addition as well.

Notifications can be sent to the most specific editors – if there is a guest editor, the managing editor will not be bothered with notifications of reviews being submitted or reviewers declining. However, all editors in the chain going back to the administrator

will have the authority to make changes to the system (i.e., submit transactions to operate on child nodes of the section they are assigned to).

4.13.2 Interface Protocol

4.13.2.1 PutRecord

ODL-Review Parameters:

transaction
XML fragment corresponding to a transaction.

XOAI Parameter Encoding:

metadataPrefix
odl_review

metadata
transaction

XOAI Request Encoding:

```
...Verb=PutRecord&metadataPrefix=odl_review&metadata=<odl_rev  
iew><actor>oai_reviewuser_2<%2Factor><resource><addse  
ction><name>Volume%20<%2Fname><section>oai_review_1<%  
2Fsection><%2Faddsection><%2Fresource><%2Fodl_review>
```

4.13.2.2 ListMetadataFormats

ODL-Review Results:

odl_review

4.13.2.3 ListSets

ODL-Review Results:

Not supported.

4.13.2.4 GetRecord

ODL-Review Parameters:

resource_identifier
OAI identifier of the resource for which to list a summary.

XOAI Parameter Encoding:

identifier
resource_identifier

metadataPrefix
resource_summary

4.13.2.5 ListIdentifiers

Inherited from XOAI-PMH / OAI-PMH.

4.13.2.6 ListRecords

ODL-Review Parameters:

report_type
Type of the report requested:
editing_summary: list of resources for which the user is an editor.
reviewing_summary: list of resources for which the user is a reviewer.
children_summary: list of children for a given resource.
open_summary: list of all sections open for submission.
notclosed_summary: list of all sections open for submission or reviewing.
submitted_summary: list of all resources submitted by a user.

resource_identifier or user_identifier
OAI identifier of the resource or user for which to generate the report.

start
Index of first item to return from complete list. This, along with the next parameter, allows selection of a range of results from within the complete list.

stop
Index of last item to return from complete list.

XOAI Parameter Encoding:

metadataPrefix
report_type

set
start/stop/resource_identifier
start/stop/user_identifier

XOAI Request Encoding:

```
...Verb=ListRecords&set=start/stop/resource_identifier&meta-  
dataPrefix=report_type...
```

Additional ODL-Review Results:

hits
Estimated total number of hits.

XOAI Response Encoding:

```
...  
<responseContainer>  
  <hits>hits</hits>  
</responseContainer>  
...
```

4.13.3 Transaction Formats

4.13.3.1 Assign editor to section or resource

Parameters:

- actor
Identifier of the editor performing the assignment (from the user database).
- role
Name of the editorial role.
- name
Name of the editor.
- id
Identifier of the editor being assigned.
- resource
Identifier of the transaction corresponding to the creation of the section or resource to which the editor is assigned.
- deadline
Date by which the reviewing must be completed.

Sample XML encoding:

```
<odl_review>
  <actor>oai:reviewuser:1</actor>
  <editor>
    <assign>
      <role>Managing Editor</role>
      <name>Boots Cassel</name>
      <id>oai:reviewuser:2</id>
      <resource>oai:review:1</resource>
      <deadline>2002-12-12</deadline>
    </assign>
  </editor>
</odl_review>
```

4.13.3.2 Decline to be the editor for a resource or section

Parameters:

- actor
Identifier of the editor who is declining.
- id
Identifier of the assignment transaction.

Sample XML encoding:

```
<odl_review>
  <actor>oai:reviewuser:2</actor>
  <editor>
    <decline>
      <id>oai:review:2</id>
    </decline>
  </editor>
</odl_review>
```

4.13.3.3 Revoke editorial privileges from an editor

Parameters:

- actor
Identifier of the editor who is revoking the privilege.
- id
Identifier of the assignment transaction.

Sample XML encoding:

```
<odl_review>
  <actor>oai:reviewuser:1</actor>
  <editor>
    <revoke>
      <id>oai:review:2</id>
    </revoke>
  </editor>
</odl_review>
```

4.13.3.4 Submit a resource for review

Parameters:

- actor
Identifier of the submitter.
- title
Name of the resource.
- baseURL
URL of the OAI-PMH archive where the resource may be found.
- id
OAI identifier of the resource.
- metadataPrefix
Metadata format in which the resource exists.
- section
Section in which to add the resource, or identifier of resource being resubmitted.

Sample XML encoding:

```
<odl_review>
  <actor>oai:reviewuser:20</actor>
  <resource>
    <submit>
      <title>Test Title #1</title>
      <baseURL>http://anarchive.org/OAI</baseURL>
      <id>oai:anarchive:123</id>
      <metadataPrefix>oai_dc</metadataPrefix>
      <section>oai:review:4</section>
    </submit>
  </resource>
</odl_review>
```

4.13.3.5 Create a new section

Parameters:

- actor
Identifier of the editor.

name
Name of the new section.

section
Existing section in which to add the new entry.

Sample XML encoding:

```
<odl_review>
  <actor>oai:reviewuser:2</actor>
  <resource>
    <addsection>
      <name>Volume 1, Issue 4</name>
      <section>oai:review:1</section>
    </addsection>
  </resource>
</odl_review>
```

4.13.3.6 Change the status of a section or resource

Parameters:

actor
Identifier of the editor.

id
Identifier of the resource or section.

status
New status for the section or resource.

Sample XML encoding:

```
<odl_review>
  <actor>oai:reviewuser:2</actor>
  <resource>
    <changestatus>
      <id>oai:review:6</id>
      <status>open</status>
    </changestatus>
  </resource>
</odl_review>
```

4.13.3.7 Move a resource or section

Parameters:

actor
Identifier of the editor.

id
Identifier of the resource or section.

section_id

New section to which to move the existing resource or section.

Sample XML encoding:

```
<odl_review>
  <actor>oai:reviewuser:2</actor>
  <resource>
    <move>
      <id>oai:review:6</id>
      <section_id>oai:review:9</section_id>
    </move>
  </resource>
</odl_review>
```

4.13.3.8 Submit a review

Parameters:

actor

Identifier of the reviewer.

baseURL

URL of the OAI-PMH archive where the review has been stored.

id

OAI identifier of the review.

metadataPrefix

Metadata format of the review.

resource_id

Internal identifier of the resource being reviewed.

Sample XML encoding:

```
<odl_review>
  <actor>oai:reviewuser:7</actor>
  <resource>
    <review>
      <baseURL>http://reviews.org/OAI</baseURL>
      <id>oai:reviewmd:458</id>
      <metadataPrefix>odl_review_2</metadataPrefix>
      <resource_id>oai:review:12</resource_id>
    </review>
  </resource>
</odl_review>
```

4.13.4 Report Formats

4.13.4.1 Resource Summary

The following is a sample metadata record generated in response to a **GetRecord** request with a *metadataPrefix* of “resource_summary”. Most fields correspond to those of the transactions listed in the previous sections.

```

<section>
  <editor>
    <role>Administrator</role>
    <id>oai:reviewuser:1</id>
  </editor>
  <section>
    <title>JERIC</title>
    <internal_id>oai:review:1</internal_id>
    <editor>
      <role>Managing Editor</role>
      <name>Boots Cassel</name>
      <id>oai:reviewuser:2</id>
      <internal_id>oai:review:2</internal_id>
    </editor>
  </section>
  <section>
    <title>Volume 1 Issue 4</title>
    <internal_id>oai:review:4</internal_id>
    <editor>
      <role>Guest Editor</role>
      <name>Bill Yurcik</name>
      <id>oai:reviewuser:4</id>
      <internal_id>oai:review:5</internal_id>
    </editor>
    <resource>
      <title>Test Title #1</title>
      <internal_id>oai:review:6</internal_id>
      <status>underreview</status>
      <submitter>oai:reviewuser:34</submitter>
      <baseURL>http://anarchive.org/OAI</baseURL>
      <id>oai:anarchive:123</id>
      <metadataPrefix>oai_dc</metadataPrefix>
      <editor>
        <role>Resource Editor</role>
        <name>JAN Lee</name>
        <id>oai:reviewuser:5</id>
        <internal_id>oai:review:7</internal_id>
        <deadline>2002-12-12</deadline>
      </editor>
      <editor>
        <role>Reviewer</role>
        <name>Hussein Suleman</name>
        <id>oai:reviewuser:6</id>
        <internal_id>oai:review:8</internal_id>
        <deadline>2002-12-09</deadline>
      </editor>
    </resource>
    <review>
      <internal_id>oai:review:9</internal_id>
      <baseURL>http://reviews.org/OAI</baseURL>
      <id>oai:reviewmd:456</id>
      <metadataPrefix>odl_review_1</metadataPrefix>
      <editor>oai:review:8</editor>
    </review>
    <draft>
      <internal_id>oai:review:9</internal_id>
      <title>Old title for test #1</title>
      <baseURL>http://reviews.org/OAI</baseURL>
    </draft>
  </section>
</section>

```

```

        <id>oai:reviewmd:456</id>
        <metadataPrefix>odl_review_1</metadataPrefix>
    </draft>
</resource>
</section>
</section>
</section>

```

4.13.4.2 Editing Summary

The following is a sample metadata record generated in response to **ListRecords** with a *metadataPrefix* of “editing_summary”. Most fields correspond to those of the transactions listed in the previous sections. *path* is a text field that indicates the position of the resource within the tree of sections as a list from the root node to the parent node of the resource.

```

<resource>
  <title>Test Title #1</title>
  <internal_id>oai:review:1</internal_id>
  <editor>
    <role>Administrator</role>
    <deadline>2002-12-09</deadline>
  </editor>
  <path>JERIC / Volume 1 / Section 1</path>
</resource>

```

4.13.4.3 Reviewing Summary

The following is a sample metadata record generated in response to **ListRecords** with a *metadataPrefix* of “reviewing_summary”. The semantics of individual fields are the same as for the “editing_summary” case.

```

<resource>
  <title>Test Title #1</title>
  <internal_id>oai:review:1</internal_id>
  <editor>
    <role>Administrator</role>
    <id>oai:reviewuser:1</id>
    <deadline>2002-12-09</deadline>
  </editor>
  <path>JERIC / Volume 1 / Section 1</path>
</resource>

```

4.13.4.4 Submitted Summary

The following is a sample metadata record generated in response to **ListRecords** with a *metadataPrefix* of “submitted_summary”. The semantics of individual fields are the same as for the “editing_summary” case.

```

<resource>
  <title>Test Title #1</title>
  <internal_id>oai:review:1</internal_id>
  <path>JERIC / Volume 1 / Section 1</path>
  <status>underreview</status>
</resource>

```

4.13.4.5 Children Summary

The following is a sample metadata record generated in response to **ListRecords** with a *metadataPrefix* of “children_summary”. The semantics of individual fields are the same as for the “editing_summary” case.

```

<section>
  <title>Section 1</title>
  <path>JERIC / Volume 1</path>
  <internal_id>oai:review:3</internal_id>
</section>

```

4.13.4.6 Open Summary

The following is a sample metadata record generated in response to **ListRecords** with a *metadataPrefix* of “open_summary”. The semantics of individual fields are the same as for the “editing_summary” case.

```

<section>
  <title>Section 1</title>
  <internal_id>oai:review:3</internal_id>
  <path>JERIC / Volume 1</path>
</section>

```

4.13.4.7 NotClosed Summary

The following is a sample metadata record generated in response to **ListRecords** with a *metadataPrefix* of “notclosed_summary”. The semantics of individual fields are the same as for the “editing_summary” case.

```

<section>
  <title>Section 1</title>
  <internal_id>oai:review:3</internal_id>
  <path>JERIC / Volume 1</path>
</section>

```

4.13.5 Interoperability Issues

The stream of transactions can be harvested to duplicate the functionality on a peer component. Also, this stream can be used as an audit trail to indicate the history of operations performed on a single resource. To enable this functionality, a component must store set information to link transactions to the resources upon which they operate.

4.14 CASE STUDY: USING AN ODL-SEARCH COMPONENT

Figure 4.4 illustrates a sequence of interactions corresponding to a typical use of an ODL-Search-compliant component. The ODL network consists of a source of data in the form of an OAI-compliant archive and a component that understands ODL-Search. The user interface layer is made up of a client's Web browser and the Web server, with scripts to generate HTML pages and forward requests to the ODL network.

There are two functions performed: indexing of the data and searching. In the former case, the ODL-Search-compliant component harvests data from the source archive using a typical harvesting algorithm, such as periodic **ListRecords** requests with the date range used to obtain only new or updated records.

To perform a search, the user submits a query by filling in a form on an HTML page. This query is then sent to the Web server, which invokes a script (or handler) to process it. The script extracts the parameters, formulates an ODL-Search **ListRecords** request and submits this to the ODL-Search-compliant component. Upon receiving the request, the component performs a search using its internal indices and then proceeds to obtain each metadata record from the source OAI archive. The metadata records are merged together and returned to the script as a single **ListRecords** response. The script then formats this response for display and it is sent back to the user in the form of a "search results" HTML page.

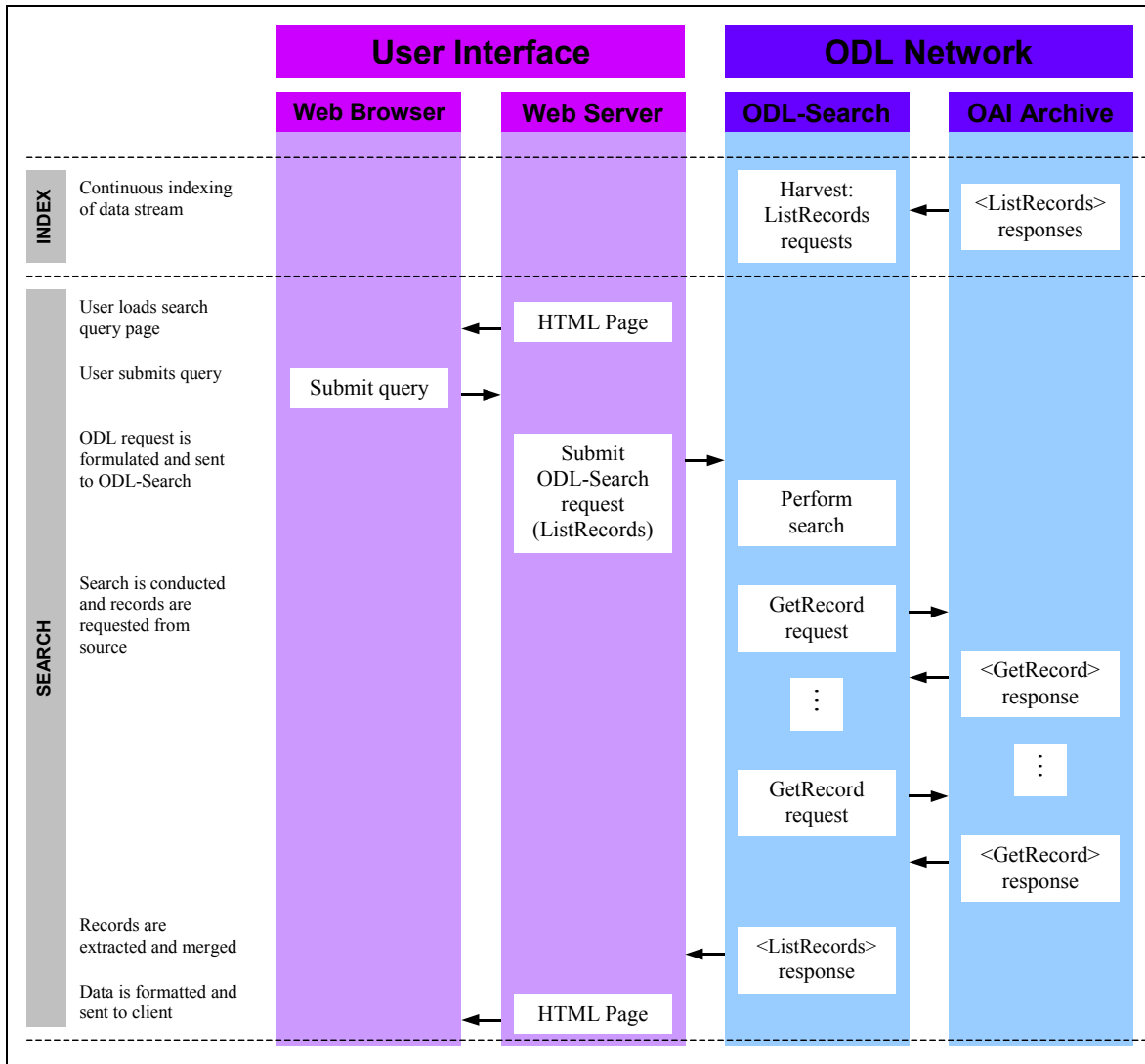


Figure 4.4 Interface and component interaction during indexing and search operations

Chapter 5

IMPLEMENTATION AND CASE STUDIES

5.1 INTRODUCTION

In order to test the feasibility of the proposed componentized architecture for digital libraries using real world scenarios, various components were implemented to support basic DL services, using ODL and OAI protocols for both inter-component communication and communication between the components and the user interface.

These components were then integrated into multiple digital library systems with different requirements and varied technologies. This chapter discusses approaches and issues in implementation and integration. That is followed in the next chapter by testing techniques and then by analysis and evaluation in the following chapter.

The implemented components can be considered as reference implementations of the previously discussed ODL protocols. Table 5.1 lists the components, brief descriptions of them, and the protocols that they respond to.

Name of Component	Description	Interface Protocol
DBUnion	Multiple data source merger	ODL-Union
IRDB	Search engine	ODL-Search
DBBrowse	Category-based browser	ODL-Browse
WhatsNew	Tracker for recent entries	ODL-Recent
Box	Dumb archive supporting submit and retrieve	ODL-Submit
Thread	Threaded annotation engine	ODL-Annotate
Suggest	Recommender system	ODL-Recommend
DBRate	Ratings manager	ODL-Rate
DBReview	Peer review workflow manager	ODL-Review

Table 5.1 ODL reference components, descriptions, and protocols

In addition, some OAI components were created to assist in setting up OAI repositories at remote locations and to standardize the implementation of OAI repositories for different projects. Unlike the ODL components that are aimed at provision of services, OAI components implement data provider interfaces for archives. These are listed, along with their descriptions, in Table 5.2.

Name of Component	Description
ETD-db Extensions	Adds OAI-PMH v1.1 support to the ETD-db software for management of ETD workflow.
XMLFile	Creates a standalone OAI-PMH v2.0 data provider to expose a system of files as an OAI archive.
Filter	Filters an OAI archive to correct errors in implementation such as errors with XML encodings.

Table 5.2 Names and descriptions of OAI components

Initially, for the prototype ETD Union Catalog system (Suleman and Fox, 2001), some ODL components were derived from the original OAI protocol rather than the XOAI protocol, to allow for the use of existing testing and validation tools. Subsequent to the construction of the ETD Union Catalog system, all components were upgraded to conform to the relevant ODL protocols.

5.2 IMPLEMENTATION

5.2.1 Platform

Each component was built as a separate set of scripts using the Perl language, with data stored in MySQL databases and files. At first, components used standard Perl modules such as XML::DOM to process XML documents using the DOM interface (Apparao, et al., 1998). These modules are distributed through the Comprehensive Perl Archive Network so the modules are reasonably portable – this was tested in the second case study. However, many of the modules had pre-requisites and some used external libraries that did not always install successfully. To minimize component installation problems arising because of this, some of the core Perl modules were rewritten in pure Perl. This involves modules to perform CGI string processing, XML parsing, and DOM tree manipulation.

All coding was done on a Linux platform, with some testing on Digital Unix and Microsoft Windows as well. Wherever possible object-oriented programming was adopted to maximize the use of common components, e.g., modules that assist in harvesting and publishing data through OAI interfaces. Some components also were sub-classed from others (e.g., DBReview from Box) – this is discussed later.

5.2.2 Customization

The configurable information for each component is stored in an appropriate configuration file, in a well-defined XML format specific to each component. This configuration is used to define the archives from which to harvest data as well as the harvesting parameters, the databases to use and how to access them as well as any parameters needed for the internal operation of the component.

Configuration is done by means of a command-line configuration script that reads in the parameters from the XML file and poses a sequence of questions to the user, with

additional supporting explanations as appropriate. The scripts are specific to each component and, in addition to setting variables, also perform sanity tests to check that:

- The database access modules, if necessary, are installed properly.
- The database, if specified, is accessible.
- The OAI/ODL source archive, if specified, is accessible.
- The *metadataPrefix* and *set* parameters are legal for any source OAI/ODL archives.
- The formats for various other component-specific parameters are legal.

During this configuration process, each component is created as an instance of the associated component template or class, with its own interface script and data storage area. The program code for the component resides in libraries shared with other instances of the same component template. This prevents unnecessary duplication of code. In addition, where a project uses multiple components, all shared libraries can be merged to further optimize space use, while sacrificing some component independence in favor of greater cohesion. Figure 5.1 shows an example of the directory structure used for components to achieve this flexibility.

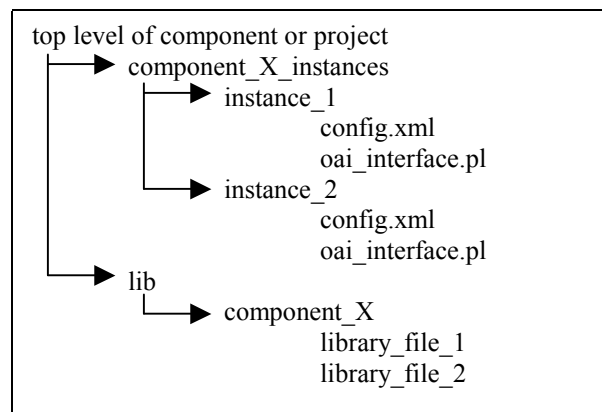


Figure 5.1 Directory layout for a typical component

5.2.3 Component Details

5.2.3.1 DBUnion

DBUnion aggregates metadata from multiple Open Archives and ODL components into a single collection, itself accessible through an XOAI-PMH interface. The internal architecture of the DBUnion component is illustrated in Figure 5.2. There are two major parts to the component: the first uses an OAI/ODL harvester to collect metadata from various sources and store it in a database; the second makes this metadata available through an XOAI-PMH interface. The harvester part of the component is periodically called by a *cron* scheduler rather than staying resident in memory.

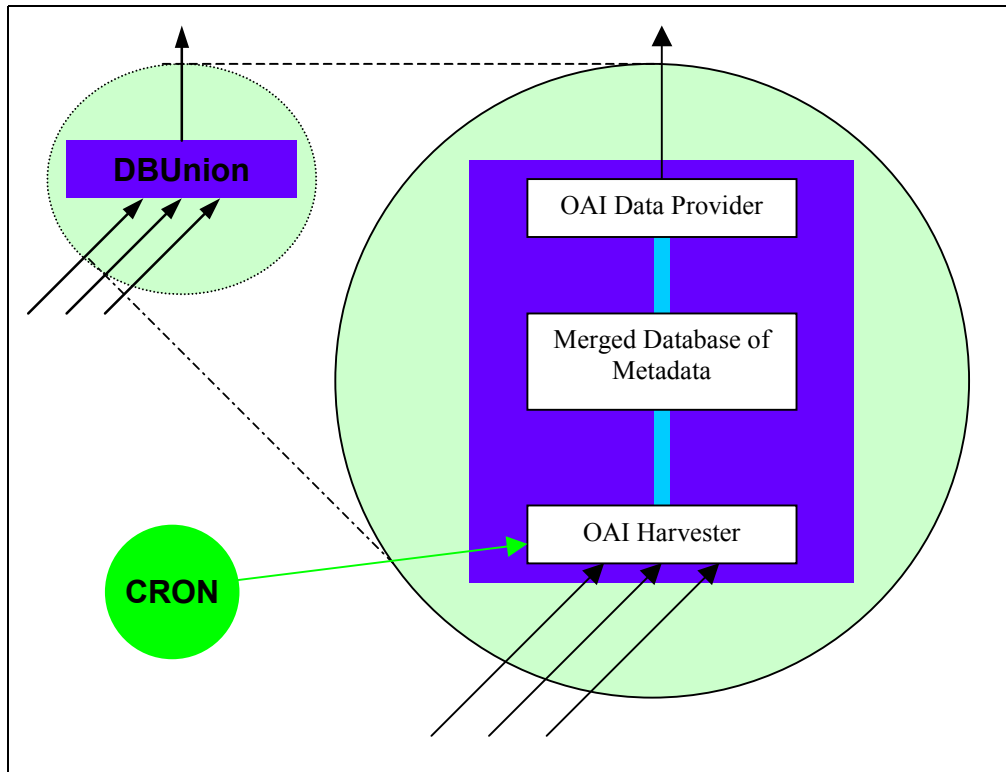


Figure 5.2 Internal architecture of DBUnion

The configuration for DBUnion is specified using an XML file. An example of this is shown in Figure 5.3. The parameters are mostly free-form text fields, but using XML allows for infinite levels of multiplicity and nesting. In the example shown, there are three metadata sources being harvested and the data is stored in the specified database.

The database is specified by its name, a username and password to access it, and a table prefix to use for this instance of the component. This allows for the use of different databases and also for multiple components or multiple instances of a single component using the same database, while avoiding naming conflicts.

```

<unionconfig>

  <database>DBI:mysql:etdunion</database>
  <dbusername>etdunion</dbusername>
  <dbpassword></dbpassword>
  <table>unioncat</table>

  <archive>
    <identifier>VTETD</identifier>
    <url>http://oai.dlib.vt.edu/cgi-bin/VTETD/VTETD.pl</url>
    <metadataPrefix>oai_dc</metadataPrefix>
    <interval>0.25</interval>
    <interrequestgap>15</interrequestgap>
  </archive>

  <archive>
    <identifier>VTETD</identifier>
    <url>http://oai.dlib.vt.edu/cgi-bin/VTETD/VTETD.pl</url>
    <metadataPrefix>oai_etdms</metadataPrefix>
    <interval>0.25</interval>
    <interrequestgap>15</interrequestgap>
  </archive>

  <archive>
    <identifier>HUBerlin</identifier>
    <url>http://dochost.rz.hu-berlin.de/OAI-script</url>
    <metadataPrefix>oai_dc</metadataPrefix>
    <interval>1</interval>
    <set>HUBerlin:dissertationen</set>
  </archive>

</unionconfig>

```

Figure 5.3 Sample configuration for DBUnion

The parameters for each archive are explained in Table 5.3. There can be multiple *archive* definitions and each may be associated with multiple metadata formats. For consistency, when multiple metadata formats are being used, the algorithm for harvesting uses a different sequence of OAI requests, which are slower and consume more network bandwidth. To avoid the extra overhead where consistency is not absolutely required, multiple *archive* definitions with different *metadataPrefix* parameters can be specified for a single baseURL.

Parameter	Syntax	Semantics	[M]andatory, [R]epeatable
identifier	String of characters	A unique identifier for the repository.	M
url	Full URL	BaseURL of archive.	M
metadataPrefix	Registered metadataPrefix	Metadata format or formats to harvest.	MR
interval	Integer	Frequency of harvesting in seconds.	M
interrequestgap	Integer	Number of seconds to wait between sending back “resumptionToken”s.	(defaults to 15)
overlap	Integer	Number of seconds to overlap when doing incremental harvesting.	(defaults to 172800)
set	setSpec	The single set to harvest from as opposed to the whole collection.	(defaults to whole collection – i.e., no set)
granularity	‘second’ or ‘day’	Whether to use timestamps with or without time information, respectively.	(defaults to ‘second’)

Table 5.3 Parameters for DBUnion

5.2.3.2 IRDB

IRDB is a simple search engine – it collects metadata from a single OAI/ODL source and pre-processes it in order to support search queries submitted through its own ODL-Search interface. The search engine uses a database to store and index inverted file data. This database is then used to support answering of term-based queries. The architecture of IRDB is shown in Figure 5.4.

The parameters supplied during configuration are very similar to those for the DBUnion component with the exception that multiple archives are disallowed. This restriction exists in order to avoid having to store the origin of each record as well – with just a single source it can be assumed that this is always the origin. The need for an originating archive arises because IRDB does not store copies of the original metadata. Instead, it processes the metadata and discards it, assuming that if the metadata is required in the future the component can always reissue a **GetRecord** request to the source archive. Thus, when a search is conducted through **ListRecords**, the set of identifiers is determined from the inverted files and then metadata for each is obtained from the source archive and merged before being returned to the requestor.

At the core of the component is a simple IR (information retrieval) system, written expressly for this purpose in Perl. The IR module indexes arbitrary fragments of XML by extracting individual words (after removing stopwords and stemming) and using tags to denote fields and sub-fields. The query language is “odlsearch1”, as specified in the previous chapter, and the ranking algorithm uses a simple sum of individual weights. Based on the popular observance that most users do not browse through all search results (Nielsen, 2001), the IR system only retrieves the n highest-ranking documents for each term in order to provide up to n ranked results. This lazy evaluation technique, suggested by techniques used in the MARIAN system (France, 2001), greatly reduces search time

but still provides the same results to users (albeit without an accurate estimate of the size of the full result set).

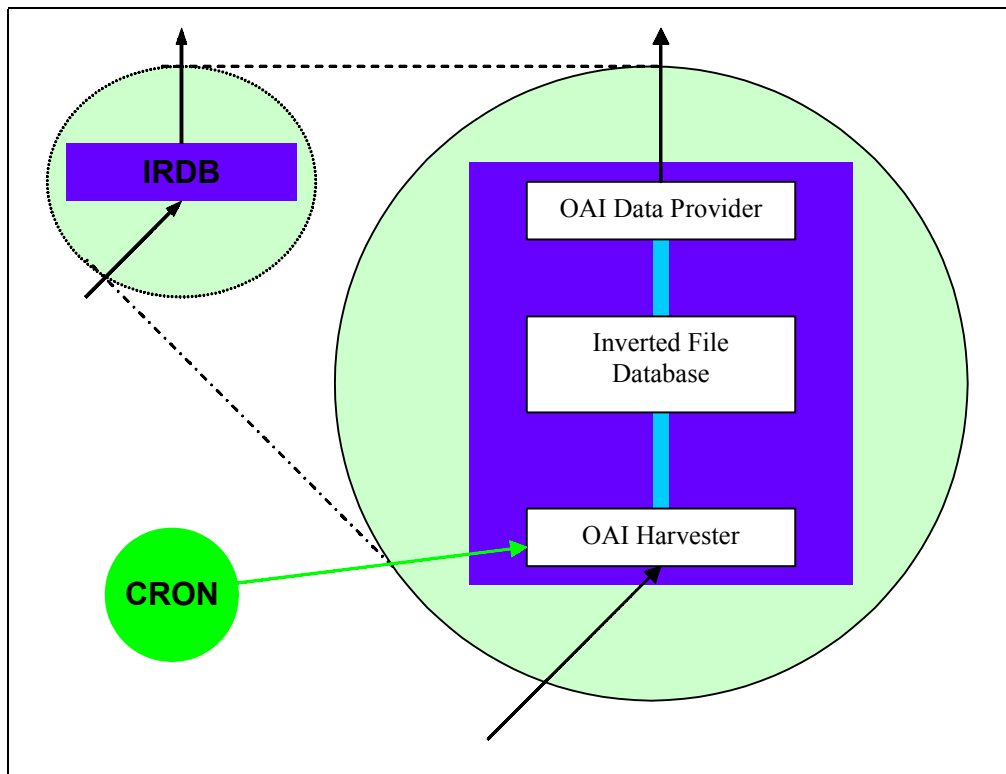


Figure 5.4 Internal architecture of IRDB

5.2.3.3 DBBrowse

DBBrowse indexes records by particular fields within the metadata to support category-based browsing. It also supports sorting of the results by arbitrary pre-specified fields. The internal architecture is similar to IRDB but the tables that are stored are indices for individual fields. The ODL-Browse protocol is used to submit requests for records to the component – these are then translated into appropriate SQL queries to extract identifiers from the pre-calculated index tables.

Configuration information for DBBrowse is a superset of that for DBUnion, with the addition of a specification of fields that need to be indexed. Figure 5.5 shows a sample fragment from a typical configuration file, and Table 5.4 discusses the syntax and semantics of these specifications.

The DBBrowse component accepts queries in the “odlbrowse1” format, as specified in the previous chapter.

```

<browser>
  <name>year</name>
  <field>dc/date</field>
  <type>controlled</type>
  <retransform>
    <from>^(.*)((0-9){4})(.*)$</from>
    <to>$2</to>
  </retransform>
</browser>

<browser>
  <name>title</name>
  <field>dc/title</field>
  <type>freetext</type>
</browser>

```

Figure 5.5 Fragment of DBBrowse configuration

Parameter	Syntax	Semantics	[M]andatory, [R]repeatable
name	String of characters	The name by which the field will be accessed by the component.	M
field	XPath (Clark and DeRose, 1999) specification	The position of the text to index within each XML record.	MR
type	'controlled' or 'freetext'	Whether the field can be used for restricting the search space or just for sorting, respectively.	M
retransform	Containing <from> and <to> fields	Regular expression transformations for controlled vocabulary fields. Once one "from" field matches, the "to" transformation is applied and subsequent "retransform"s are ignored.	R

Table 5.4 Parameters for DBBrowse

5.2.3.4 WhatsNew

WhatsNew, implementing the ODL-Recent protocol, provides a random sample of recent records encountered by the component when harvesting from its source archive. It stores all recent identifiers in a database table and when a request for records is encountered, it extracts a fixed number of these at random and sends back either the identifiers or the corresponding metadata records, depending on the request.

The configuration information is similar to DBUnion and the internal architecture is similar to IRDB. One additional configuration parameter, *recententries*, specifies the number of items to return in response to **ListIdentifiers** or **ListRecords**.

5.2.3.5 Box

The Box component implements ODL-Submit, a simple extension of the XOAI protocol to support adding records to an archive. The Box component stores arbitrary fragments

of XML in a database keyed on the *identifier* and *metadataPrefix* parameters. There is no predetermined metadata format supported by the component – it simply determines the list of formats by extracting the relevant information from the *xsi:schemaLocation* attribute that is found in the root element of every OAI record. Sets can be specified using the *sets* parameter of **PutRecord** and these are indexed in a separate table to support per-set harvesting.

Configuration of the component involves specifying the database in terms of its location and the username and password to use to connect to it. Since this component is an archive that others can harvest from, additional OAI-like parameters are specified, e.g., the number of records to generate before creating a *resumptionToken*. To prevent unauthorized changes to the component, access control lists are specified to indicate, by IP address or domain name, which machines are allowed to read from and write to the component.

Box supports two additional external interfaces: a simplified OAI interface to make the component behave like an OAI-compliant repository (as opposed to an XOAI-compliant component) and a rudimentary user interface to support viewing and editing of records stored in the component. Figure 5.6 displays the front page of this user interface for a typical archive.

The screenshot shows a web browser window with the title "Box Archive Editor". Below the title, it displays the "baseURL of Box" as "[http://oai.dlib.vt.edu:80/~hussein/cgi-bin/ODL-Box/Box/test1/box.pl]". A horizontal line separates this from a paragraph of text: "This is an administrative interface to browse through the contents of a Box archive and add/modify/delete metadata records. This is NOT a user interface to the Box - such should be provided by a service provider that uses the Box component." Another horizontal line follows. Below this, a prompt reads: "Select a service request to start with (enter parameters if necessary):". There are four buttons: "Put Record", "List Sets", "List Identifiers", and "Get Record". The "List Identifiers" button is followed by input fields for "from", "until", "set", and "resTok". The "Get Record" button is followed by input fields for "id" and "mdp". At the bottom left is a link to "Home", and at the bottom right is the text "a part of the Open Digital Libraries project".

Figure 5.6 Direct editing interface to Box component

This user interface uses just the XOAI service requests in order to communicate with the archive. By issuing a sequence of **ListIdentifiers**, **ListMetadataFormats**, **GetRecord**, and **PutRecord** commands, it is possible to directly edit the contents of the archive for testing and maintenance purposes.

Such an interface is naturally a security hazard, so an additional access control list is maintained to specify which client machines will be allowed access to the editing functionality.

5.2.3.6 Thread

Thread implements the ODL-Annotate protocol to store and retrieve threaded annotations. New annotations are stored in a database and requests for annotations are retrieved when necessary.

Threading of messages and replies is accomplished by means of an automatically generated thread number associated with each entry. This thread number is used to order the entries so they resemble a discussion forum. New entries are given a thread number higher than the last entry in the system. Replies to previous entries are given a thread number mid-way between the two existing entries above and below the reply to be inserted. If the thread numbers above and below the entry are consecutive, then thread numbers for all entries from the topmost up until the point of insertion are renumbered so that gaps are created for the current and future operations. This process is illustrated in Figure 5.7.

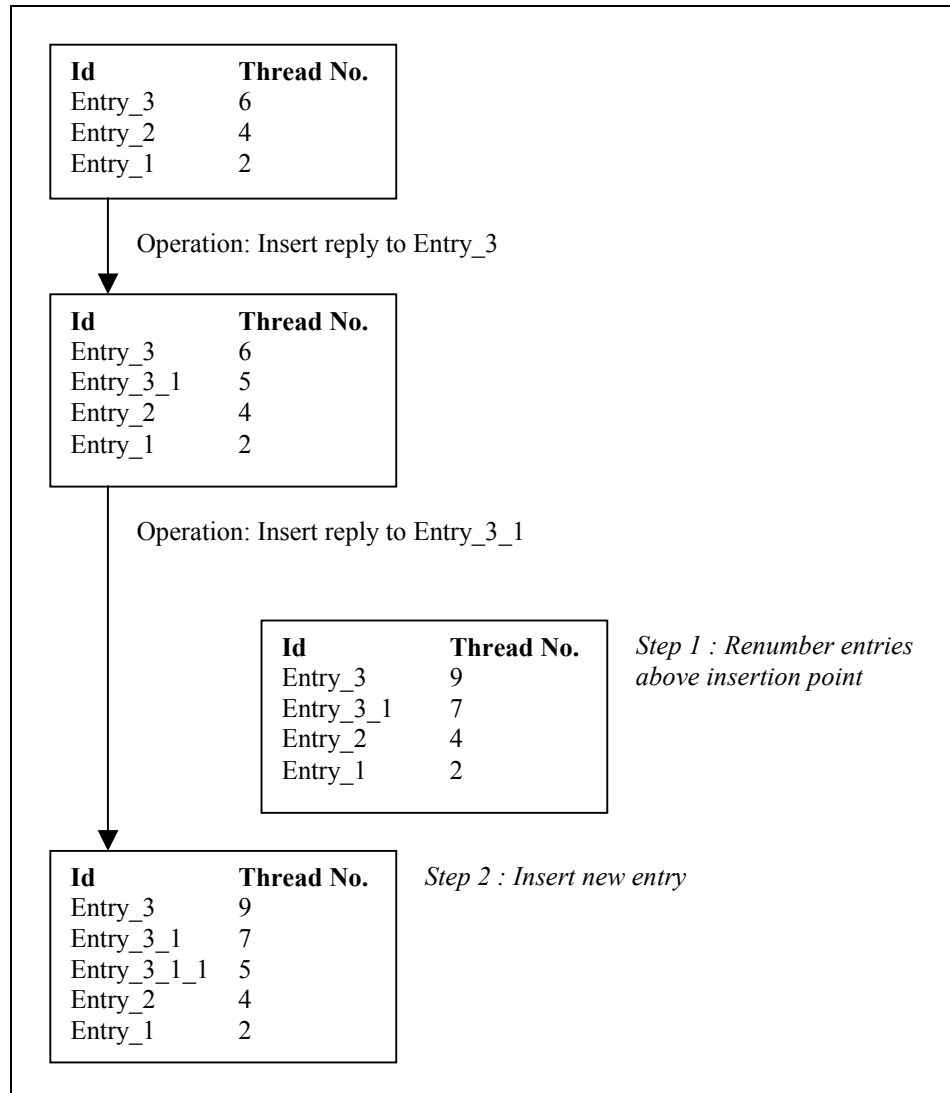


Figure 5.7 Insertion procedure for new entries in a Thread component

Configuration of Thread requires the specification of a database in terms of location and connection details. In addition, the threading epoch can be specified – where epoch refers to the maximum (initial) difference in thread numbers. This epoch is used to assign new thread numbers as well as expand the “space” between two entries if needed. Larger epoch values result in fewer rescaling operations, which are database intensive. However, the maximum number of annotations that may be stored in the component depends on the highest thread number so unnecessary gaps are undesirable. For instances with few replies (and therefore little actual threading), small epoch numbers are more suitable, hence this option is configurable on a per-instance basis.

5.2.3.7 Suggest

Suggest implements the ODL-Recommend protocol to provide recommendations to users based on past history of activity. Each time a user accesses a resource, this interaction is

submitted to the Suggest component and stored in a database. In this context access refers to any activity that constitutes a basis for recommendation, typically the viewing of metadata or the digital object. Requests for recommendations are then generated from this list of stored interactions.

The algorithms used to generate recommendations use only the information stored in the (user, resource) tuples stored by the component, as listed below:

To generate a list of users who have accessed a given resource R:

- Return all users with resource=R

To generate a list of users with similar interests to a given user U:

- Find all resources AR with user=U

- For each resource R in AR

- Find all users AU who accessed R

- For each user U in AU

- Add a vote to U in AU2

- Return the users from AU2 with the highest votes

To generate a list of resources accessed by users who had accessed a given resource R:

- Find all users AU with resource=R

- For each user U in AU

- Find all resources AR accessed by user U

- For each resource R in AR

- Add a vote to R in AR2

- Return the resources from AR2 with the highest votes

To generate a list of resources accessed by users with similar interests to a given user:

- Find all resources AR with user=U

- For each resource R in AR

- Find all users AU who accessed R

- For each user U in AU

- Add a vote to U in AU2

- Select into AU3 the users from AU2 with the highest votes

- For each user U in AU3

- Find all resources AR2 accessed by user U

- For each resource R in AR2

Add a vote to R in AR3

Return the resources from AR3 with the highest votes

Configuration is similar to the Box component, including the specification of access control lists to allow access from only a pre-specified group of machines. The following parameters must be specified to restrict the number of iterations of the recommendation algorithms:

- The maximum number of recent entries for a user to use as an indication of that user's interests.
- The maximum number of votes the most popular item ought to receive before the algorithm stops iterating.
- The maximum number of items recommended before the algorithm stops generating new items.

5.2.3.8 DBRate

DBRate implements the ODL-Rate protocol to specify ratings and return average ratings for resources. The ratings are stored in a database where each entry contains a resource identifier, a user identifier, and a rating value. User identifiers are considered to be unique keys, thus avoiding a single user submitting multiple ratings for a single item – this also allows users to submit new ratings which automatically overwrite the previous values.

DBRate was sub-classed from the Box component and thus stores all ratings as simple XML records. Requests for average ratings result in retrieval and processing of all relevant records to dynamically determine the average value. This method is not the most efficient in terms of space and time, but allowed for faster development based on prior modules.

Configuration requires just the specification of a database, and database connection details.

5.2.3.9 DBReview

DBReview manages the workflow of a peer-review system and interfaces with the user interface via the ODL-Review protocol. Database tables are used to store entries for sections/resources, editors, and draft submissions. Transactions submitted to the system are stored in a Box (from which the component was sub-classed) and then parsed and stored in the appropriate tables.

Wherever necessary, operations are preceded by checks to ensure that the user has the relevant editor status to perform the operation. Resources and sections are stored in reverse order of insertion, using the same threading approach devised for the Thread component.

All requests for reports are then satisfied by interrogating the database tables and generating appropriate XML records.

An earlier version of the component stored workflow management information in an XML file instead of a database. This file was parsed prior to each operation and DOM was used to extract information, insert new entries, and prune the tree as necessary to satisfy queries about the workflow. This approach, while elegant from a coding perspective, was abandoned because parsing of the entire XML file was not scalable for non-trivial numbers of transactions.

DBReview was designed to be used in conjunction with other supporting components. In testing the component for the JERIC project, multiple instances of the Box component were used to store metadata, reviews, and user information. This is discussed in section 5.3.4.

5.2.4 OAI Component Details

5.2.4.1 ETD-db Extensions

To support participation in the ETD Union Catalog, an extension to the ETD-db ETD management software (Atkins, 2001) was created. This extension is a drop-in module to retrospectively add OAI-PMH v1.1 support to an archive that uses any version of the ETD-db software; the current version of ETD-db comes prepackaged with it. The ETD-db extensions are used by 7 of the 13 sites participating in the Union Catalog.

The software is written in Perl and has the same system requirements as ETD-db. Thus, there are no additional system requirements for an existing site running ETD-db to become an OAI data provider. The software need only be copied and configured, where the configuration script confirms that the databases are accessible and allows the installer to enter OAI-specific parameters such as the “repository name” and “administrator’s email”. Once all parameters are specified, the ETD collection is accessible as an OAI data provider through the web server’s CGI mechanism.

The OAI extension to ETD-db will export metadata in four different formats: Dublin Core (DCMI, 1997), ETDMS (Atkins, et al., 2001), MARC (Library of Congress, 2002b), and RFC1807 (Lasher and Cohen, 1995). For each record, the Dublin Core version is equivalent to the ETDMS version, with additional tags removed. The MARC version of the record follows the crosswalk recommended by the ETDMS specification, while the RFC1807 version is simply a best-effort mapping since no official crosswalk exists.

5.2.4.2 XMLFile

XMLFile is a module that makes a collection of files – usually in XML format and where each corresponds to a single record – into an OAI-PMH v2.0 data provider. It is meant to require a minimum of effort while retaining all the flexibility of the OAI protocol.

The component file layout and configuration follow the pattern established by the ODL components, with a clean separation between the data provider engine, configuration data, and the data being served.

Support is provided for hierarchical sets, which are mapped to the directory structure of a specified file system. Symbolic links within the file system also are taken into account and files that are symlinked are considered to be items that exist in multiple sets, as per the OAI protocol.

While source files are usually in XML format, this does not have to be the case. Source files can be in any format as long as a transformation program exists to map the source file to each of the desired metadata formats in XML. In addition, source files can be selected from a file system if they match particular regular expressions – this way metadata files embedded within a heterogeneous collection need not be separated.

5.2.4.3 Filter

Filter obtains data from one archive and transforms the requests and/or results in order to make the archive appear more conformant with the recommendations and requirements of the OAI protocol. In particular, two archives participating in the ETD Union Archive (MIT and Technical University of Dresden) do not produce correctly encoded XML and/or use a system of identifiers that is not globally unique. The Filter component is used to overcome these issues by pre-processing and post-processing all OAI requests and responses, respectively.

Unlike most other components, the Filter does not store any data locally. Thus there is no need for separate data storage facilities for different instances – the only difference is in configuration so this is done through parameters in the interface script. An example of this interface script is shown in Figure 5.8.

```
use Filter;

sub main
{
  new Filter (
    'http://theses.mit.edu/Dienst/Index/2.0/OAI-1.0',
    {
      prependid    => 'oai:MIT:',
      shortendate   => 'yes',
      cleanxml      => 'yes',
      ignorefrom    => 'yes',
      ignoreuntil   => 'yes',
    }
  );
}
```

Figure 5.8 Sample configuration script for Filter

The configurable parameters, their syntax, and their semantics are explained in Table 5.5.

Parameter	Syntax	Semantics
prependid	String of characters	Append the specified string of characters to every identifier. Conversely, remove it from identifiers that are passed in as parameters.
shortendate	yes or no	If the archive uses a more complete variant of the ISO8601 date format, truncate the additional parts.
cleanxml	yes or no	If the archive uses characters that are illegal in XML, escape or remove them.
ignorefrom	yes or no	Ignore all "from" date parameters and use entire collection.
ignoreuntil	yes or no	Ignore all "until" date parameters and use entire collection.

Table 5.5 Parameters for Filter

5.2.5 User Interfaces

5.2.5.1 XSL

It was decided from the outset that user interface design was not a priority in proving the feasibility of the ODL approach. However, simple user interfaces were needed for the experimental systems to demonstrate communication with the component layer. These user interfaces were created using direct parsing of the XML data in some instances but more often than not by transforming the XML using XSL stylesheets (Clark, 1999).

A typical user interface script contains a number of subroutines to react to different user input conditions. Each of these subroutines performs the following actions:

1. Prepare input variables for a component.
2. Submit a request to the appropriate component.
3. Check the response and transform it using an XSL stylesheet.
4. Perform cleanup and then iterate steps 2-3 as necessary for other components.

In some cases, the submission of a request is handled by the XSL stylesheet itself. XSL provides a "document" command to retrieve HTTP documents. An example of such a command in XSL is:

```
<xsl:variable name="index" select="document (concat
(suggestURL, '?verb=ListIdentifiers&set=RR/', start,
 '/', stop, '/', resource)))/xoaili:ListIdentifiers"/>
```

This corresponds to the ODL command:

```
...verb=ListIdentifiers&set=RR/start/stop/resource
```

In the XSL example, the variable "index" is set to point to the root node of the XML response to **ListIdentifiers** that is sent to the Suggest component identified by the *suggestURL* parameter. In this manner, a stylesheet applied to one document can

internally formulate HTTP requests and retrieve additional documents. After retrieving a list of identifiers, as shown above, a stylesheet can iterate over the identifiers and retrieve individual records using **GetRecord** without leaving the stylesheet.

Another technique to make the stylesheet submit service requests is to format one or more baseURLs into an XML fragment, together with supporting parameters, and submit this to the stylesheet. The stylesheet then can combine these baseURLs and parameters into HTTP requests, obtain the associated responses, and format the XML fragments for display. As an example of this, the new CSTC system uses the following Perl statement to submit parameters to a stylesheet to generate the feedback display for an item based on responses from a Thread component:

```
$xslt2->Transform("<forum>".  
"<baseURL>$ODLCSTCFeedback</baseURL>".  
"<identifier>$identifier</identifier>".  
"<start>$start</start>".  
"<size>$size</size>".  
"<fidentifier>$fidentifier</fidentifier>".  
"</forum>");
```

The baseURL of the Thread component is indicated by the *\$ODLCSTCFeedback* variable. *\$identifier* is the identifier of the resource that is displayed. *\$fidentifier* is the identifier of the currently displayed annotation. *\$start* and *\$size* indicate the range of replies to the current annotation to be displayed.

The stylesheet that processes this XML fragment submits multiple ODL-Annotate requests for the following:

- The metadata record for the annotation being displayed.
- The list of metadata records for replies to the currently displayed annotation.
- The parent record of the currently display annotation.

These three responses are then transformed into a single fragment of HTML to resemble a discussion forum associated with each resource.

5.2.5.2 MEdit: Schema-based Metadata Editor

Editing of metadata is traditionally a non-trivial task when performed over the Web because HTML user interfaces are fixed while metadata editing requires a greater degree of flexibility. A typical problem occurs when a metadata field is repeatable – there is no simple way to duplicate a single field in a static HTML form.

Some preliminary work was done on creating input forms dynamically based on XML Schema specifications to better support the editing of metadata. This approach gracefully deals with problems such as the repeatability of individual fields, as well as the arbitrary levels of nesting that are allowed in XML-encoded metadata.

MDEdit, the Perl module implementation resulting from this work, interprets XML Schema files and CGI variables and produces an HTML input form that may be further manipulated. Its features include:

- Generalized support for nested metadata fields.
- Multiplicity of fields at all levels of the metadata (where the metadata is hierarchically organized).
- Automatic checks for minimum occurrences of fields.
- Supports features of HTML input forms using XSD <appinfo> extensions.
- File uploading to associate files with URLs in metadata.

MDEdit understands a subset of the W3C's XML Schema language. Thus, while every schema supported by MDEdit is a valid XSD, the reverse does not hold true. The following sections describe the special features of schemata used by MDEdit.

Types

The supported type constructs are:

- string
- complexType, containing a sequence
- simpleType, containing a restriction with enumerations

The following are therefore valid type definitions:

```
<element name="test" type="string"/>
```

```
<element name="test">
  <complexType>
    <sequence>
      <element name="test1" type="string"/>
      <element name="test2" type="string"/>
    </sequence>
  </complexType>
</element>
```

```

<element name="test">
  <simpleType>
    <restriction base="string">
      <enumeration value="test1"/>
      <enumeration value="test2"/>
    </restriction>
  </simpleType>
</element>

```

Type Propagation

Named types may be defined globally and used for elements and within other types. However, `<ref>`s are not supported. This should not affect the declarative power of the schema, just the way in which it is written. Only one globally-scoped element should exist (others will be ignored), that being the root element of the metadata record.

An example of using a named type is:

```

<element name="test" type="testType"/>
<complexType name="testType">
  <sequence>
    <element name="test1" type="string"/>
    <element name="test2" type="string"/>
  </sequence>
</complexType>

```

Extensions

According to the XML Schema standard, every element may have an annotation tag with an `<appinfo>` containing tags specific to the application. Table 5.6 lists the tags defined by MDEdit as application-specific extensions to the XML Schema specification.

Tag	Description
<caption>	an alternate caption to use instead of the tag name
<description>	a description of what the field is
<fixeddescription>	a description that must always be displayed
<rows>	number of rows to use for the field display
<columns>	number of columns to use for the field display
<inputtype>	'password' – password entry box; displays "*" instead of characters 'file' – file upload box 'radio' – use radio buttons for list instead of <select>
<externallist>	name of a list that is populated at run-time by the script that generates the form

Table 5.6 List of additional MEdit schema tags to define appearance of HTML forms

An example of an annotation to a schema is:

```
<element name="test" type="string">
  <annotation>
    <appinfo>
      <caption>Test Input Field</caption>
      <rows>40</rows>
    </appinfo>
  </annotation>
</element>
```

This results in an input field in the HTML form with the text label “Test Input Field” and a text box 40 characters wide for data entry/editing.

Creating and using the MEdit object

In order to use the MEdit module, the MEdit object must be created, passing in the name of the annotated schema file as a parameter. Then, the form is created based on the combination of this object and either CGI parameters or an existing XML record. When the user eventually submits the form, validation checks are performed on the metadata and if all the constraints are satisfied, an XML record can be generated. If all constraints are not satisfied, the form can be regenerated with an error message and highlighting of invalid or incomplete fields. In-between the initial form and the final XML record the user may click on “+” buttons alongside the fields to add more fields (or collections of fields).

Rendering based on schema and extensions

The HTML rendering rules are as follows:

1. All *string* fields are simple text fields if no parameters are given.
2. `<simpleType>` enumerations are lists – either `<select>` lists or radio buttons based on the `<input type="checkbox">` field.
3. `<columns>` specifies the width of text fields and `<select>` lists.
4. `<rows>` specifies the number of rows in `<select>` lists and radio buttons.
5. If `<rows>` > 1 for a *string*, then a multi-line `<textarea>` is created.
6. If `maxOccurs` > 1 for a list, then multiple options may be selected in a `<select>` list. For a *string*, this indicates that the field may be duplicated `maxOccurs` times and if there are not already `maxOccurs` fields, a “+” button is added next to the field to allow creation of additional fields.
7. If `minOccurs` > 1 for a *string*, at least `minOccurs` copies of the field are created.

Figure 5.9 shows a typical HTML rendering of a simple annotated schema from the new CSTC system. The schema used to generate this interface is listed in Appendix A.

Figure 5.9 HTML rendering of metadata editor

5.2.5.3 Content Negotiation

For the JERIC and CSTC systems, multiple data formats are used and this requires different interfaces for entry, editing, and display for each of the formats. Entry and editing of metadata in both cases uses the MEdit module, specialized by a schema file for the required metadata format.

Displaying different types of metadata can be accomplished using different XML namespaces for the different formats, while sharing a single *metadataPrefix* at the OAI

and ODL levels. For example, in the new CSTC system, a record can be obtained from the metadata repository by specifying an *identifier* and a *metadataPrefix* of “resource”. The resulting record can be in many different formats, each distinguishable only by its XML namespace. The XSL stylesheet that renders the record then invokes the appropriate template based on the namespace.

5.3 CASE STUDIES

5.3.1 Case study: ETD Union Catalog

5.3.1.1 Requirements

NDLTD is an organization comprised of multiple member universities and institutions, many maintaining collections of electronic theses and dissertations (Fox, 2002; Fox, et al., 1996; Fox, et al., 1997; Suleman, et al., 2001). These individual collections are maintained at each institution, completely separate from other efforts. One of the current projects of NDLTD involves integrating the collections from various sites so that, ultimately, researchers can access any publicly available ETD from one or more mirrored central sites. The requirements for this include the ability to find particular ETDs (e.g., search, browse) but it does not include a submission mechanism since that is already handled at the remote sites. In addition, the merged collection of metadata needs to be accessible separately from the rest of the user interface so that other user portals may be built to access the data.

5.3.1.2 Architecture

Figure 5.10 shows the architecture of ODL components that was devised in order to provide the required services using a network of components rather than a monolithic system. Each remote site operates as an OAI data provider and the data from these sites is collected periodically using a DBUnion component. This, in turn, serves as a source of data for the higher-level IRDB, DBBrowse and WhatsNew components, which are then used by the user interface to facilitate access to the underlying distributed collection.

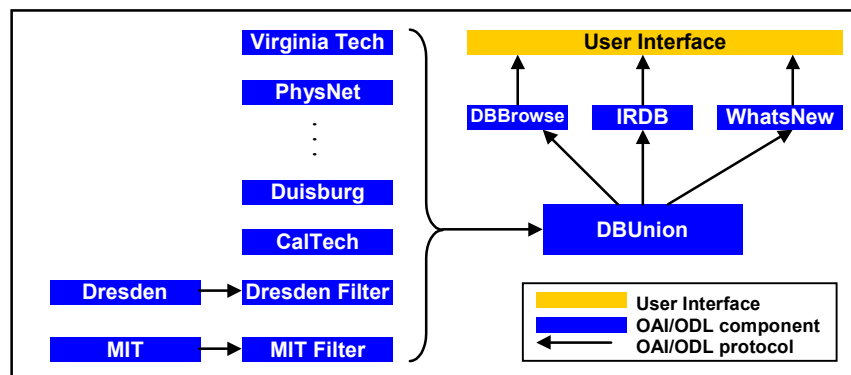


Figure 5.10 Architecture of NDLTD ODL system

5.3.1.3 Interface

The user interface uses a handful of Perl scripts to bind HTML forms to ODL component usage. XSL stylesheets are used to transform the output from various ODL components into suitable human-readable versions and links are inserted where appropriate by these stylesheets. Figure 5.11 shows the main index page for the experimental system, and Figure 5.12 shows the output from a typical browse operation.

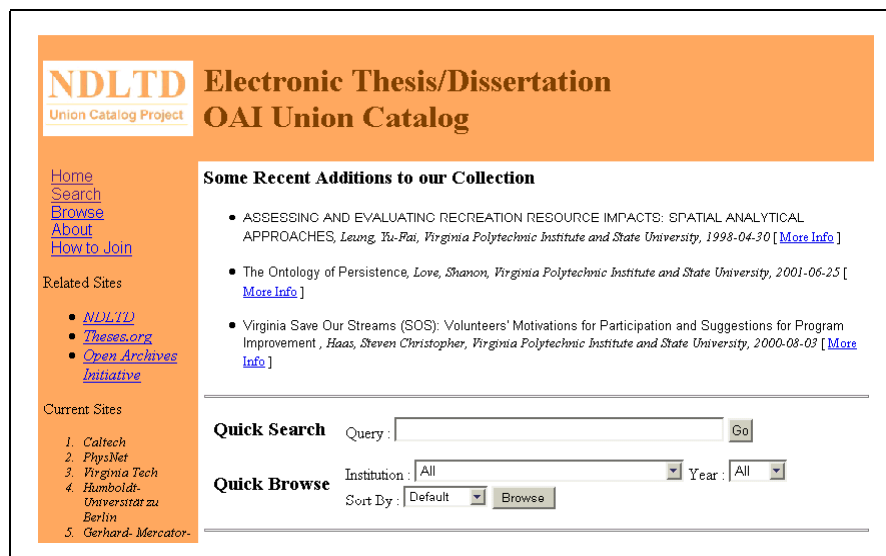


Figure 5.11 Index page for NDLTD user interface

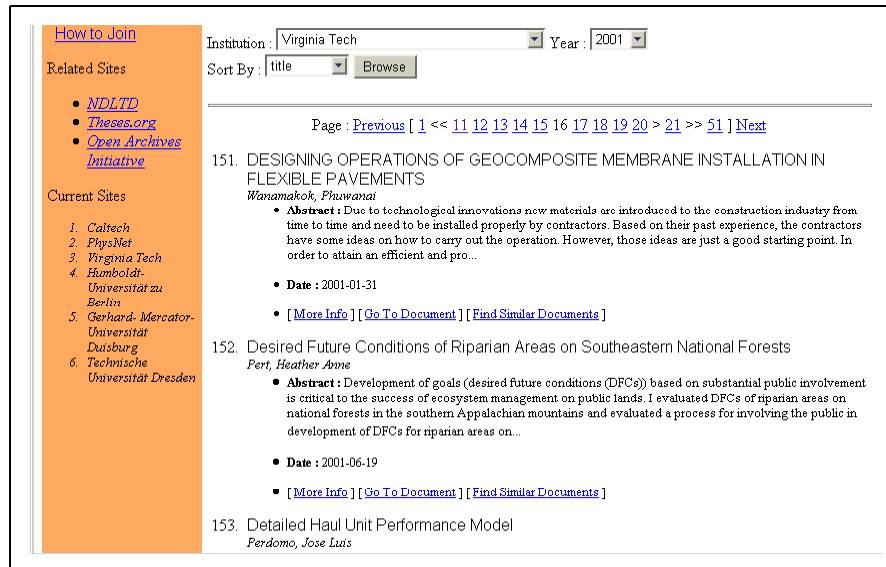


Figure 5.12 Output from typical browse operation

5.3.2 Case study: CSTC

5.3.2.1 Requirements

The Computer Science Teaching Center (Fox, et al., 2002b) is an existing production DL with the following services provided to patrons: submission, retrieval, search, browse, and peer review. Originally, all services were provided by performing SQL queries on a single mSQL database but this is not most efficient for all types of services. Specifically, the search and browse functions were served by this single database, which did not allow the viewing of subsets of results. As CSTC became more popular this resulted in scaling problems so the browsing and searching functions needed to be updated or replaced.

5.3.2.2 Architecture

Since the metadata is already accessible through an OAI interface, this was used to link the DBBrowse ODL component into CSTC. This change is only for the purpose of demonstration so just the DBBrowse component was incorporated into the CSTC system. Figure 5.13 shows the architecture of the CSTC ODL and OAI components in relation to the rest of the system.

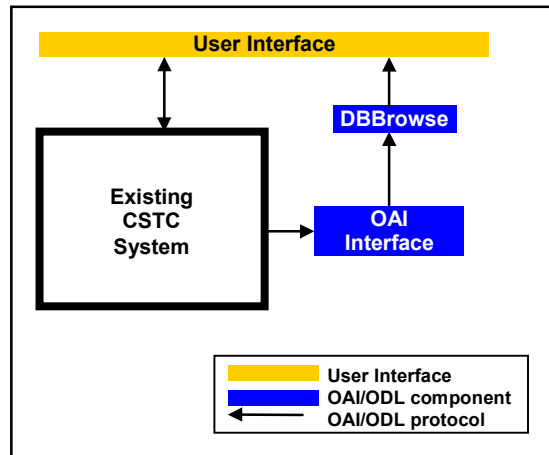


Figure 5.13 Architecture of CSTC, showing ODL components

5.3.2.3 Interface

As before, a set of Perl scripts serves as the “glue” between the user interface and the component. XSL stylesheets are used to transform the lists of records, but these are then integrated with the standard user interface elements of the existing system. Figure 5.14 shows a screen snapshot of the user interface for browsing.

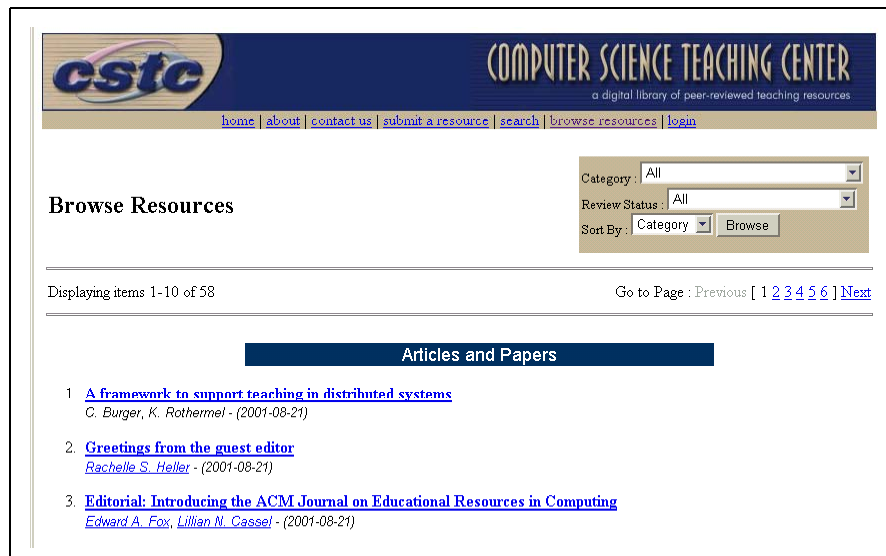


Figure 5.14 CSTC interface for browsing, using DBBrowse component

5.3.3 Case study: husseinsspace.com

5.3.3.1 Requirements

husseinsspace.com is a personal website, with a comparatively large content base, that is regularly visited by several people who are no longer in touch with one another. To encourage more communication among visitors, a guestbook – in the form of a threaded discussion forum – was installed, using an instance of the Thread component. Visitors to the website may view previous comments, reply to one of them, or add new comments. This use of the Thread component can be applied to any website where a simple feedback or discussion mechanism is desired.

5.3.3.2 Architecture

The Thread component operates independently of the rest of the site, so there is no interaction (see Figure 5.15) as in previous instances and since the entire site is the resource being annotated, there is only one set of annotations. This approach is well suited to a system where additional features are added at a later date.

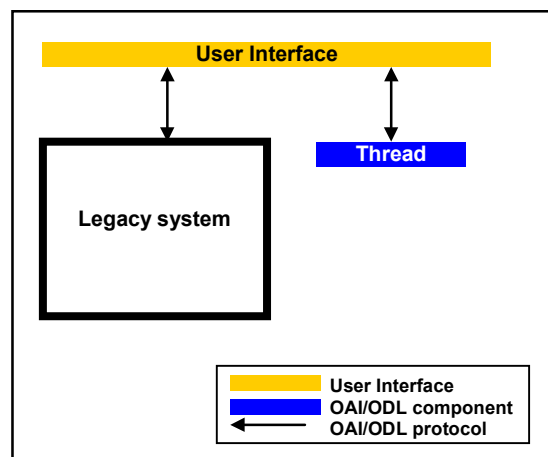


Figure 5.15 Architecture of the guestbook addition to husseinsspace.com

5.3.3.3 Interface

The look and feel of the interface is carried into the XSL stylesheets, so, while the component functions independently of the rest of the site, users perceive a cohesive system, as depicted in Figure 5.16.



Figure 5.16 User interface for the guestbook on husseinsspace.com

5.3.4 Case study: JERIC

5.3.4.1 Introduction

Background

The Computer Science Teaching Center (CSTC), as discussed previously, is a digital library of educational resources for computer science and related disciplines. From its inception, one of its major distinguishing features has been the peer reviewing of such resources, enabled by the system. This also has had the disadvantage of raising the entry requirements without providing a tangible benefit to submitters, negatively impacting the rate of submission in the early years of the project.

The Journal for Educational Resources in Computing (JERIC) (Fox and Cassel, 2002) was launched in response to a perceived need for recognition of creators of educational resources. Just like CSTC, JERIC solicits submission of educational material and subjects such submissions to a peer review process. The prime difference is that JERIC material is earmarked for publication in an ACM journal while CSTC material is available online through a free-access web interface. Further, CSTC and JERIC are inter-related:

- CSTC serves as a type of pre-print service for JERIC.
- CSTC metadata-only entries are allowed for resources published in JERIC, with the former pointing to the latter.

Since CSTC already has a web interface that allows peer review of its resources, this interface was adapted to cater to the needs of JERIC as well. Unfortunately, the transition was not smooth and it rapidly became obvious that a new and specialized reviewing system will need to be employed.

While the ACM is pursuing the acquisition of a web-based reviewing system (to be used by JERIC and other journals) for the long term, it was felt that a short term solution may

be feasible and can have the side-effects of providing an updated reviewing system for CSTC and related projects.

Evaluation of CSTC/JERIC review system

Positives:

- CSTC separates resources from reviews, storing each as a first-class metadata object; this maintains the integrity of the original resource in perpetuity.
- Reviews are split into sections for confidential and non-confidential comments (as far as the author is concerned).
- Reviews are assigned based on interests and reviewing history.

Negatives:

- CSTC uses a concept of universal roles – a user is a reviewer, an editor, or just a plain user. This does not cater for situations where a single user can be editor of an issue of the journal but a reviewer for a single submission in a different issue. In small closed systems, such as in the Wiki collaborative hypertext systems (Leuf and Cunningham, 2001), users may be able to police themselves but in CSTC this has not happened.
- CSTC has only three roles for users. In practice, journals are known to have at least “managing editors”, “assistant editors”, “guest editors”, “reviewers”, and users. Conferences have “conference chairs”, “program chairs”, “workshop chairs”, etc. The simple 3-role model does not work well in these situations.
- Users are confused by the CSTC and JERIC submissions being channeled through the same system. Also, there is no way to distinguish among CSTC and JERIC editors and reviewers, or reviewing interests related to CSTC and JERIC.
- Many queries cannot be answered by the CSTC system since there is no comprehensive system history outlining, for example, exactly which user assigned which resource to which reviewer.
- CSTC uses some older technology (e.g., login system and database access) and this does not fully support new de facto standards for accessing online information. Many users of CSTC have complained that the system does not function as they expect it to. The investment in older technology makes it more difficult to satisfy the demands that CSTC keep up with the latest and greatest ways of doing things on the Internet.
- Reviewers cannot easily decline – this is currently only possible by reviewers notifying the editor through email. This action may be problematic as well, since the editor is not easily identified.
- Only those users currently registered may be selected for reviewing or editorial duties. In practice, editors quite often prefer to recruit reviewers from outside of the existing circle, but currently, such persons need to register through the online

interface, apply to be reviewers, and only then can resources be assigned to them. This is a complicated process that has confused many editors and reviewers.

- CSTC sends out some email notification (e.g., when resources are submitted) but there can be more feedback (e.g., reminders about reviewing assignments), given that email is currently the best way to communicate with users online.
- There is only one set of criteria that apply to all resources. In practice, different types of resources are best reviewed according to different criteria, e.g., resources can have different reviewing requirements from papers.

5.3.4.2 Requirements

The basic charter was to design and build a system for online submission and reviewing of resources (papers, software, etc.).

Specifically,

- The reviewing system must be specific to individual projects – possibly using a class/instance model.
- The reviewing system must be able to produce a listing of all reviewing and editing tasks pertinent to a single user.
- The reviewing system must be able to produce on demand a history based on a log of all activity related to a resource.
- The system must assign permissions on a need-to-do basis. If a user is not an editor for a particular resource then that user should not have editorial privileges.
- User roles must be flexible to cater to different system models.
- Users involved in the reviewing process must be able to accept/decline invitations for roles and the hierarchy of higher-level editors must be transparent to them.
- It must be possible to invite new reviewers and editors and directly associate resources with them (i.e., as soon as they log in, their assignments are visible). This will greatly reduce the number of steps in the current workflow.
- Email messages must be sent out for reminders, confirmations, notifications, and, in general, to alert users when they need to interact with the online interface. Such email messages must contain direct links into relevant parts of the system.
- Reviews must not be constrained to a single type – the system must allow for editors to attach, for example, an “approved by editor” note or a “pending agreement of author” note (where resources are nominated for inclusion).
- Annotations must be accessible, depending on the role of the user (editor, reviewer, etc.).

- The system must be built of components that can easily be replaced as technology changes in the future.
- The system must use current standards in the DL community, e.g., XML, XSL, OAI-PMH, etc.
- Reviews and resources must be specified by schema that can be changed later and/or adapted to different projects or uses within the same project.
- The system must be easy to maintain.

5.3.4.3 Architecture

The reviewing system was designed to make use of multiple ODL components: an XReview component (following a prototype version of the ODL-Review protocol) and multiple Box components to store metadata for resources, reviews, and users.

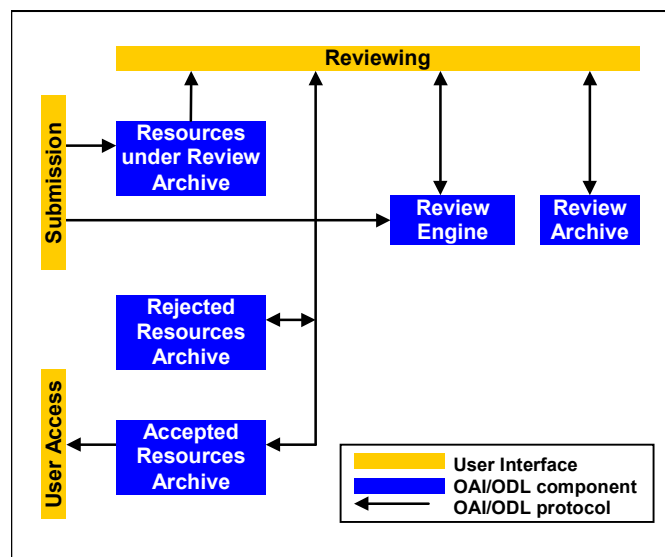


Figure 5.17 Architecture of JERIC peer review system

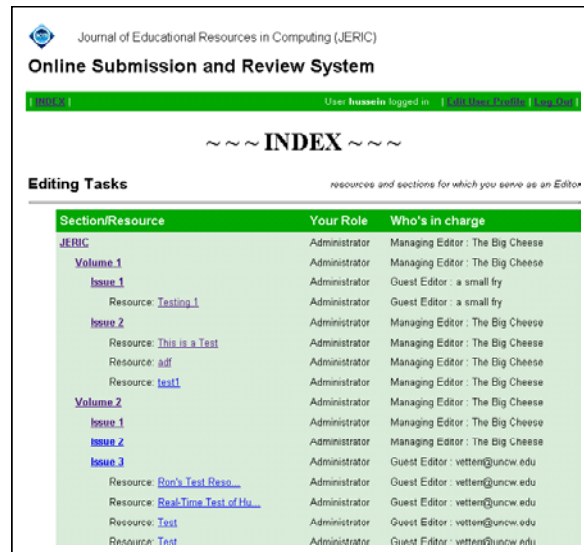
Figure 5.17 illustrates the flow of information in the reviewing system. Submissions are first made to the “resources under review” archive. To initiate the review process, the “review engine” is informed of the need to review the specified resource.

The review engine contacts the appropriate editors, who then coordinate the process of reviewing, with continuing access to the resource, reviews, and workflow information (from the review engine). Workflow information is updated as new tasks are assigned or completed during the process of reviewing the resource. Ultimately, the resource is either accepted or rejected and is then transferred to the respective final archive.

Revisions are created as new submissions. Whenever the XReview component receives a metadata object with an existing identifier, it assumes it is a revision and links it into the database appropriately.

5.3.4.4 Interface

After a user logs into the system, he or she is presented with a list of recently submitted and reviewed items. In addition, all the reviewing and editorial assignments for the user are listed. Figure 5.18 displays this opening page.



Section/Resource	Your Role	Who's in charge
JERIC	Administrator	Managing Editor : The Big Cheese
Volume 1	Administrator	Managing Editor : The Big Cheese
Issue 1	Administrator	Guest Editor : a small fry
Resource: Testing 1	Administrator	Guest Editor : a small fry
Issue 2	Administrator	Managing Editor : The Big Cheese
Resource: This is a Test	Administrator	Managing Editor : The Big Cheese
Resource: aiff	Administrator	Managing Editor : The Big Cheese
Resource: test1	Administrator	Managing Editor : The Big Cheese
Volume 2	Administrator	Managing Editor : The Big Cheese
Issue 1	Administrator	Managing Editor : The Big Cheese
Issue 2	Administrator	Managing Editor : The Big Cheese
Issue 3	Administrator	Guest Editor : vetten@uncw.edu
Resource: Don's Test Resp...	Administrator	Guest Editor : vetten@uncw.edu
Resource: Real-Time Test of Hu...	Administrator	Guest Editor : vetten@uncw.edu
Resource: Text	Administrator	Guest Editor : vetten@uncw.edu
Resource: Text	Administrator	Guest Editor : vetten@uncw.edu

Figure 5.18 User interface of JERIC peer review system

Only those resources that are pertinent to a user are displayed as active (this is determined by the stylesheet). The parent nodes are displayed to provide contextual information. Links associated with the sections and resources direct the user to pages where more information is presented. Also afforded is the ability to edit metadata/submissions/reviews where applicable.

If the user is an editor for a section or resource, then clicking on the name of the section or resource brings up a page with more information as well as the ability to view reviews, change the status of the section/resource, or view a transaction history. Figure 5.19 shows this summary view of a resource or section.

The screenshot displays a web interface titled "Issue 2". It is divided into two main sections: "Section Contents" and "Section Editing Information".

Section Contents: This section has a subtitle "list of subsections or resources within this section". It contains a table with two columns: "Sub-sections" and "Resources". The "Sub-sections" column has a "New Section" button and an "Add" button. The "Resources" column contains a link "This is a Test" and a link "add test!".

Section Editing Information: This section has a subtitle "list of editors and forms to assign those". It contains a table with two columns: "Current Status" and "Editors". The "Current Status" column has a link "Open for submission" and a link "[Close for submission] [Close for submissions/reviewing]". The "Editors" column has a link "The Big Cheese (Managing Editor, JERIC)" and a link "[Your current status = Administrator]". Below the table is a form titled "Assign New Editor:" with a "Role" dropdown menu set to "Managing Editor" and a text input field labeled "Enter email address or select from user list".

Figure 5.19 Full details for a single resource or section

Reviewers are presented with an interface to enter their reviews but not to assign any editorial staff. Users are presented with an even more restrictive interface with only the ability to view metadata and reviews (if the resource was already completely reviewed).

5.3.5 Case study: New CSTC

5.3.5.1 Requirements

The Computer Science Teaching Center was developed, starting in 1998, to collect and disseminate high quality resources for teaching and learning in computing-related topics. As a digital library system, it provides services to search and browse through resources, as well as manage the submission and reviewing of resources. While the DBBrowse component has long since been integrated into the CSTC system (see earlier section), it is hypothesized that the system as a whole can be rebuilt using ODL components to enable further modularity in the future and to support newer standards and techniques in the construction of digital libraries, such as the archival policies encouraged by the OAI.

The aim of rebuilding the CSTC is to demonstrate how multiple ODL components can be integrated seamlessly into a single online information management system.

5.3.5.2 Architecture

The design of the new CSTC is a natural extension of the JERIC system since they share a similar approach to reviewing of resources. Figure 5.20 provides an overview of this design. With the introduction of the IRDB and DBBrowse components, discovery services are provided. Using an intermediate DBUnion component enables easy importing of metadata from external sources such as the existing CSTC system. DBRate, Suggest, and Thread provide additional services that are not present in the original version: rating, recommendation, and annotation of resources.

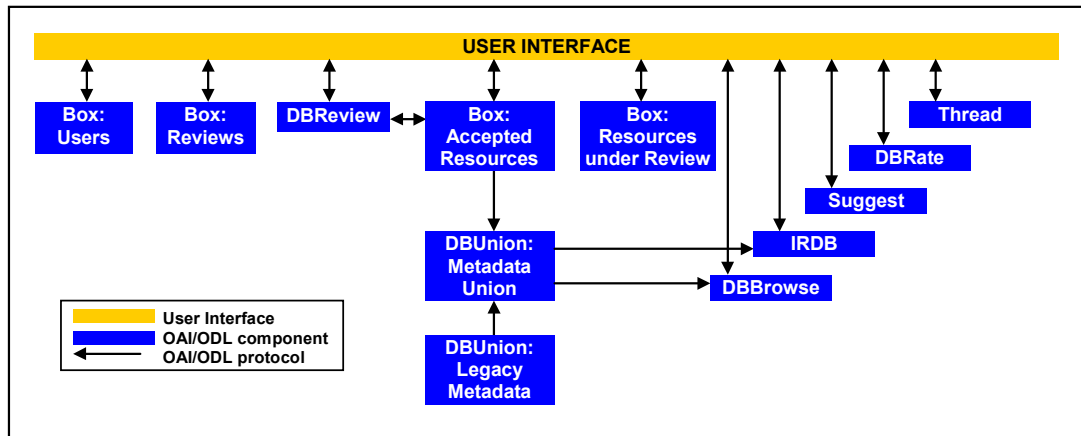


Figure 5.20 Architecture of new CSTC system

5.3.5.3 Interface

The initial welcome screen in Figure 5.21 is designed to look as similar to the existing system as possible, while incorporating the ability to register and log in at the very outset – a feature that is commonplace in many existing websites.

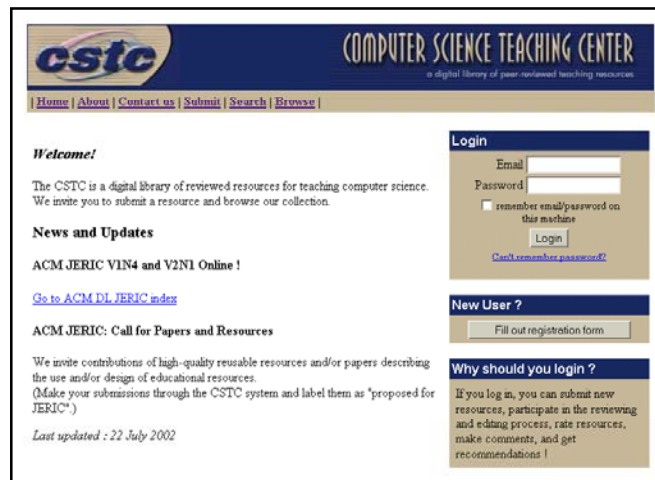


Figure 5.21 New CSTC initial welcome screen

After logging in a user is presented with the list of editing, reviewing, and submission tasks relevant to that user, as shown in Figure 5.22. New submissions can be made and previous submissions can be checked for status and availability of reviews.

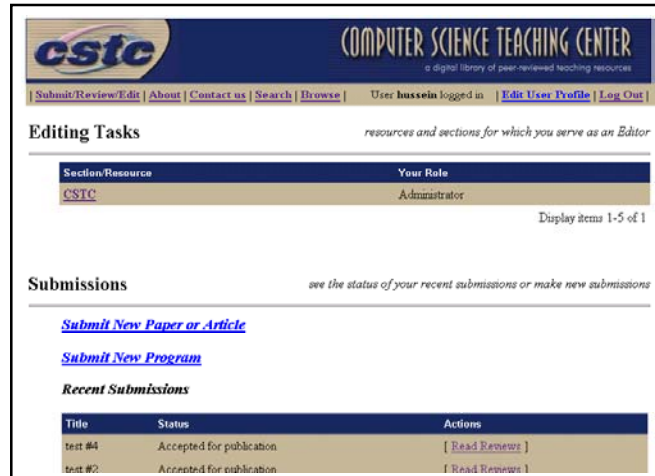


Figure 5.22 New CSTC list of editing, reviewing, and submission tasks

After resources have been reviewed, they are accessible to the search and browse functions, as shown in Figure 5.23. The user may page through the search results and select any items that they are interested in.



Figure 5.23 New CSTC resource browsing

Figure 5.24 shows a typical metadata display for a single resource. The top of the screen displays the basic metadata that was submitted along with the resource. This is followed by a display of the current rating of the resource and an interface to submit a rating. Then there are recommendations for resources that were viewed by users who had viewed this resource. Lastly, there is a threaded discussion forum specific to this resource for ongoing discussion.

CSTC Resource	
Title	Simulation of the 2-dimensional DCT
Author	Holger Wens
Author	Claudia Schrenner
Date	2001-10-29
Category	Multimedia
Files	http://www.cstc.org/data/resources/100/resource.jar Description : resource.jar contains the applet Java classes of the applet http://www-mm.informatik.uni-mannheim.de/veranstaltungen/animation/multimedia/2d_dct/ Description :

Rating
Current Rating : 7.5 (based on 2 vote(s)) ★★★★★
Add your rating :
<input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="radio"/> 6 <input type="radio"/> 7 <input type="radio"/> 8 <input type="radio"/> 9 <input type="radio"/> 10 <input type="button" value="Submit"/>

Recommendations
People who viewed this resource also viewed the following: <ul style="list-style-type: none"> Turing Machine Applet CSTC Resource: Ajae Sommer, Bernd Wierlmes - (2001-09-06) Computing Education: A Journey Continued CSTC Resource: Julian Cassel - (2001-09-06) Rotating Cube Applet Performance Test CSTC Resource: Bernd Wierlmes - (2001-09-06)

Feedback
[feedback index] [post feedback]
message index » see more -- hussein -- Mon Oct 7 09:26:37 2002 » testing -- hussein@vt.edu -- Sat Oct 5 01:26:12 2002 » Re: testing -- hussein -- Sat Oct 5 01:33:16 2002
post message subject : <input type="text"/> comments : <input type="text"/> <input type="button" value="post message"/>

© 2002 Virginia Tech DLRL

Figure 5.24 New CSTC full metadata and associated information

5.4 SUMMARY

Various components were created as reference implementations of the protocols discussed in the previous chapter, and to assist in setting up OAI data providers.

These were then configured and integrated in different ways to both extend existing systems and form the basis for complete digital library systems.

Chapter 6

COMPONENT TESTING

6.1 INTRODUCTION

Component testing can be carried out at different levels. For example, glass-box testing verifies the internal logic of a component while black-box testing verifies correct operation (when accessed through well-defined interfaces). In the case of ODL, the former corresponds to testing of the component logic code while the latter corresponds to testing of the ODL/XOAI interface. In addition, the correct operation of the component when accessed over the Web through HTTP can be tested at a slightly higher layer. These different approaches to testing are depicted in Figure 6.1 and discussed in detail in this chapter.

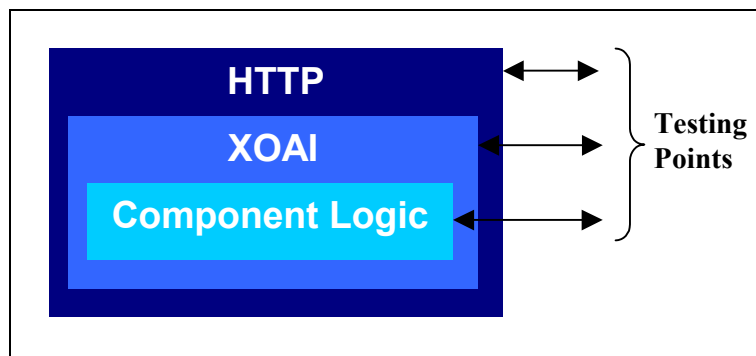


Figure 6.1 Layers at which component testing can be performed

6.2 DIRECT COMPONENT LOGIC

Some components have separable component logic and interface modules, thus allowing the component logic to be invoked by mechanisms other than the appropriate ODL interface layer. This also supports testing of the component logic without going through the ODL layer and can assist in resolving the reasons for failure during component installation and configuration. Some of the components that have such interfaces available are IRDB and DBBrowse.

6.2.1 IRDB

IRDB contains a *testsearch* utility that allows a user to submit search queries directly to the information retrieval (IR) engine from the operating system command-line and get back a set of identifiers corresponding to the matches. The ODL-Search protocol is

completely bypassed and the IR engine is instead loaded directly by the *testsearch* script. A sample invocation of this looks like:

```
./testsearch.pl 'computer science testing' 1 10
```

In this example, the query string is “computer science testing” and a result set beginning at match with rank 1 and ending at match with rank 10 (at the most) is requested. The following are the results of this operation, including a list of the identifiers for each match and an estimate of the total number of matches for the query (i.e., 90).

```
Test Search
List of identifiers in result set:
  oai:VTETD:etd-052499-165749
  oai:VTETD:etd-05142001-144644
  oai:UUdiva:1198
  oai:UUdiva:46
  oai:VTETD:etd-31598-144937
  oai:VTETD:etd-2698-12129
  oai:VTETD:etd-1598-132027
  oai:VTETD:etd-06252002-173958
  oai:VTETD:etd-05082001-010142
  oai:UUdiva:233
Total hits : 90
```

6.2.2 DBBrowse

DBBrowse contains a *testbrowse* utility that submits browse queries to the engine and lists the results. An example of such a command-line request is:

```
./testbrowse.pl 'sort(title)' 1 10
```

The results for such an operation, as displayed below, are similar to those generated for the IRDB component.

```
Test Browse
List of identifiers in result set:
  oai:CSTC:157
  oai:CSTC:139
  oai:CSTC:108
  oai:CSTC:46
  oai:CSTC:47
  oai:CSTC:141
  oai:CSTC:261
  oai:CSTC:113
  oai:CSTC:125
  oai:CSTC:264
Total hits : 79
```

6.3 XOAI INTERFACE

After confirming that the component logic is correct, the next step is to verify that the ODL layer works as expected. This entails connecting to the component through the ODL layer, but without involving the Web server.

Since all ODL components are Web applications, there must exist a mechanism to communicate between the Web server and the component for the purposes of application invocation and data transfer. In the reference implementations discussed in the previous chapter, components were created using the CGI specification (Gundavaram, 1996). The Web server spawns CGI applications whenever a corresponding URL is requested. Data is transferred between the Web server and the CGI application using a combination of standard input and setting of environment variables. In the reverse direction, data is transferred from the CGI application to the Web server using standard output. Thus, if the input conditions can be simulated, a CGI application can be run from the command-line without the need for the Web server layer. Output from the CGI application will then be sent to the standard output device – usually the controlling terminal.

ODL components use a common CGI library that accepts parameters from either standard input or environment variables. For testing purposes either can be used to simulate the operation of a Web server, but preference is given to the latter because temporary environment variables can be specified on the command-line when using shells such as *bash*.

An example of conducting such a test from the command-line is:

```
QUERY_STRING='verb=ListIdentifiers&set=odlsearch1/computer%
20science%20testing/1/10' ./search.pl
```

In this request, the search terms are structured according to the ODL-Search protocol and embedded within an XOAI-PMH request. Spaces are escaped according to the CGI specification; if an HTTP client is being used, this might occur automatically. QUERY_STRING is the name/value pair expected by *search.pl*, which is the actual ODL interface program that is run from the command-line. The results from such a request look like:


```

Content-type: text/xml

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

<xoai:ListIdentifiers>

<responseDate>2002-10-17T16:36:56-04:00</responseDate>

<requestURL>http://server.name:80/scriptname?verb=ListIdentifiers&set=odlsearch1/computer%20science%20testing/1/10</requestURL>

<identifier>oai:VTETD:etd-052499-165749</identifier>
<identifier>oai:VTETD:etd-05142001-144644</identifier>
<identifier>oai:UUdiva:1198</identifier>
<identifier>oai:UUdiva:46</identifier>
<identifier>oai:VTETD:etd-31598-144937</identifier>
<identifier>oai:VTETD:etd-2698-12129</identifier>
<identifier>oai:VTETD:etd-1598-132027</identifier>
<identifier>oai:VTETD:etd-06252002-173958</identifier>
<identifier>oai:VTETD:etd-05082001-010142</identifier>
<identifier>oai:UUdiva:233</identifier>

<xoai:responseContainer>
  <hits>90</hits>
</xoai:responseContainer>
</xoai:ListIdentifiers>

```

(XML namespace and schema information has been removed for clarity.)

The structure of the response follows the ODL-Search protocol (and, so, the OAI protocol). As expected, the sequence of identifiers as well as the value of *hits* matches the output of the previous direct component logic test.

The first line of the response contains a command that is normally sent to the Web server to indicate the MIME type of the response. This is crucial for most Web servers and is usually interpreted by the Web server and then used in formatting the headers of the response, but it does not appear in the responses for higher level tests.

6.4 WEB CLIENT TEST

At the highest layer of testing, an HTTP client can be used to submit requests and obtain results. This involves sending a request through the client to the Web server and getting a response from the Web server. Two popular clients for such tests are:

lynx: a text-mode Web browser, with the ability to simply download responses and also to report on HTTP status errors that occur. The latter is useful as ODL errors are reported through the HTTP status code mechanism used by OAI-PMH v1.1. An example of submitting a request using *lynx* is:

```
lynx -source "http://spare06.dlib.vt.edu/~hussein/cgi-  
bin/ODL-IRDB-1.1/IRDB/irdb/search.pl?verb=ListIdentifiers  
&set=odlsearch1/computer%20science%20testing/1/10"
```

wget: a file downloading utility which is not as complex as *lynx* and has a smaller memory footprint and faster execution time (see next chapter). An example of a request submitted through *wget* is:

```
wget -O - "http://spare06.dlib.vt.edu/~hussein/cgi-bin/ODL-  
IRDB-1.1/IRDB/irdb/search.pl?verb=ListIdentifiers&set=  
odlsearch1/computer%20science%20testing/1/10"
```

In both cases, since the request is channeled through the Web server, the request also must include the full baseURL that is used as the general ODL interface to the component. Carrying out this test ensures that the Web server is properly configured to execute CGI applications, with the correct permissions to perform the relevant tasks and generate the expected output. This avoids problems, such as with some older Web servers which defaulted to executing CGI applications in a very restrictive environment, so while output was generated it was not always sensible.

6.5 PARSING TEST

While the previous test obtains responses successfully from a Web server, it does not check if the responses are formatted correctly. Since all ODL responses are in the XML format, an XML parser can be used to check if the responses are well formed and can be parsed. Also, after parsing, responses can be displayed in a more structured manner, thus making it simpler to visually inspect the data.

Common Web browsers such as Internet Explorer (v5.0 and above) and Netscape Navigator (v6.0 and above) support XML as a file format that can be downloaded and displayed. Internet Explorer has a particularly useful interface where nested tags can be hidden or exposed by clicking on a “+” or “-” sign next to the containing tag. In both browsers, XML is displayed indented, with color-coding to differentiate among tags, attributes, and content. An example of an ODL request and its response in Internet Explorer is shown in Figure 6.2.

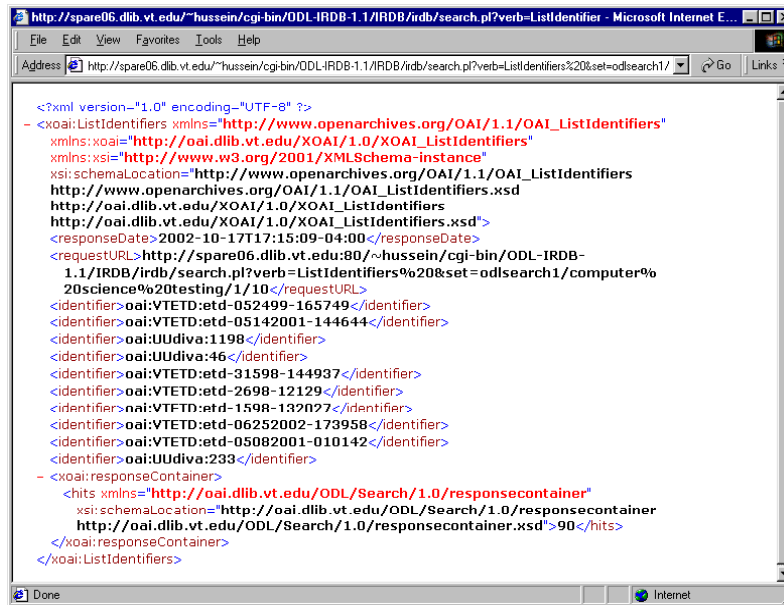


Figure 6.2 Internet Explorer displaying an ODL request and response

An additional advantage of using a Web browser is that since most developers do not actually work on the server's console or install server-side software on their desktop machines, testing a server using a client-side browser results in distinct separation between client and server. All previous tests were conducted on the server.

6.6 REPOSITORY EXPLORER

As the final step in testing, after it has successfully been determined that

- the component logic is correct,
- the ODL interface appears to work properly,
- the Web server correctly communicates with the ODL component, and
- the response appears correct and is parsable,

rigorous tests can be run to verify that the communications protocol supported by the component is being rigidly adhered to. These tests were developed initially as the Repository Explorer (Suleman, 2001; Suleman, 2002), a general tool for OAI repositories, and then generalized for ODL components in the form of the Component Explorer.

6.6.1 Introduction

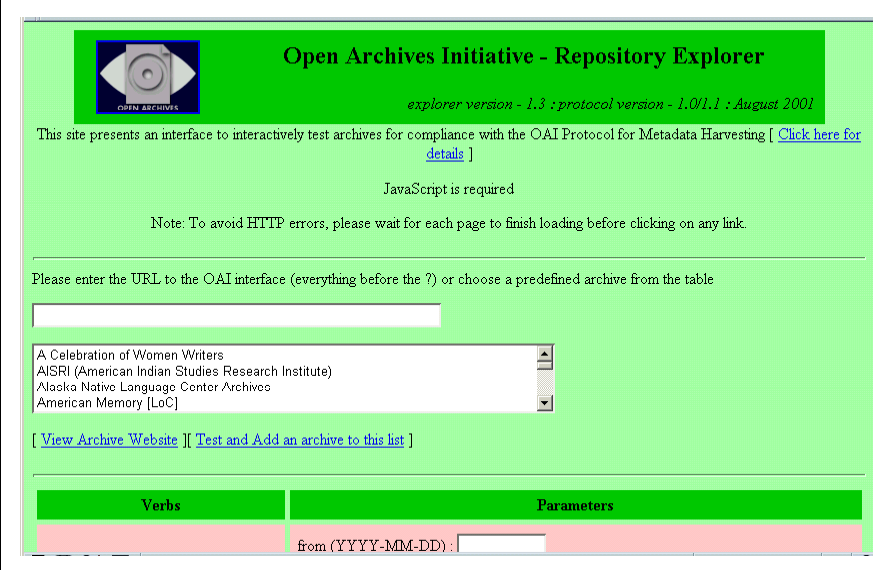
From the very outset, proponents of the OAI-PMH realized that the success of such a standard requires vigilance in specification of the protocol as well as standardization of implementation. The lack of standardized implementation is a substantial barrier to interoperability in many existing client/server protocols.

After the inaugural meeting of the OAI in 1999, a handful of archivists began to implement the agreed-upon interoperability protocol at their distributed sites. This effort was immediately hampered by a varying interpretation of the protocol specification. This was largely due to the difficulty of precisely specifying a protocol that would be both general and applicable to multiple domains. The client/server architecture chosen by the OAI led to a classic “chicken-and-egg” problem since client implementations need to interface correctly with server implementations. Subsequently, there was a low degree of confidence in the correctness of early implementations in each category. Coupled with this, even when clients and servers subscribed to the same interpretation, there was not high confidence that other client/server pairs would interoperate successfully.

As one approach to address these concerns, a protocol tester was developed to allow a user to perform low-level protocol tests on a data provider (server) implementation without the need for a corresponding service provider (client) implementation. This now widely used software, the Repository Explorer, aids in standardizing the protocol understood by various different archives subscribing to the OAI model of interoperability. This tool has a significant impact on simplifying development of interoperability interfaces and increasing the level of confidence of early adopters of the technology, thus exemplifying the positive impact of exhaustive testing and quality assurance on interoperability ventures.

6.6.2 Design of the Repository Explorer

The Repository Explorer is implemented as a web-based application (see Figure 6.3) to take advantage of the ubiquitous nature of WWW clients, and to alleviate the need to install multiple packages on client machines to support all the software used during testing.



Open Archives Initiative - Repository Explorer
explorer version - 1.3 : protocol version - 1.0/1.1 : August 2001

This site presents an interface to interactively test archives for compliance with the OAI Protocol for Metadata Harvesting [[Click here for details](#)]

JavaScript is required

Note: To avoid HTTP errors, please wait for each page to finish loading before clicking on any link.

Please enter the URL to the OAI interface (everything before the ?) or choose a predefined archive from the table

A Celebration of Women Writers
AISRI (American Indian Studies Research Institute)
Alaska Native Language Center Archives
American Memory [LoC]

[[View Archive Website](#)] [[Test and Add an archive to this list](#)]

Verbs	Parameters
from (YYYY-MM-DD) :	<input type="text"/>

Figure 6.3 Repository Explorer basic interface

The software supports both manual and automatic testing, but with an emphasis on the former. In automatic testing mode, a series of protocol requests, with legal and illegal combinations of parameters, are issued to the archive being tested, and the responses are checked for compliance with the expected range of responses. In manual mode, the software allows a user to browse through the contents of the archive using only the well-defined interface provided by the protocol. The user has full control over all parameters and can test individual features of the protocol and/or implementation.

In the manual mode of operation, users may specify a list of parameters and then select a verb from the list provided (see Figure 6.4). The request is formatted and sent to the server. The associated XML response is then validated, parsed, and displayed in a more human-readable form. Additionally, obvious choices for further browsing are converted into hyperlinks, e.g., after **ListSets** is issued, each set name from the response is converted into a link that results in **ListIdentifiers** being submitted. This is depicted in Figure 6.5.

Verbs	Parameters	
Identify List Metadata Formats List Sets List Identifiers List Records Get Record	from (eg, YYYY-MM-DD) : <input type="text"/> until (eg, YYYY-MM-DD) : <input type="text"/> metadataPrefix : <input type="text"/> identifier : <input type="text"/> set : <input type="text"/> resumptionToken : <input type="text"/>	
Language	Display	Schema Validation
English <input type="button" value="v"/>	<input checked="" type="radio"/> Parsed <input type="radio"/> Raw XML <input type="radio"/> Both	<input type="radio"/> None <input checked="" type="radio"/> Local mirror of schemata (Xerces) <input type="radio"/> Online schemata (Xerces) <input type="radio"/> Local mirror of schemata (XSV) <input type="radio"/> Online schemata (XSV)

Figure 6.4 Repository Explorer verb and parameter entry

<http://rocky.dlib.vt.edu/~jedlpix/cgi-bin/OAI2.0/hspics/jcdl/oai.pl?verb=ListSets>
 Archive details : <http://www.jcdl.org>

List of Sets

Click on the link to list the contents

[JCDL Day Four](#)

set description:
 dc:
 description: Pictures taken during JCDL Day Four

[JCDL Banquet](#)

set description:
 dc:
 description: Pictures taken during JCDL Banquet

[JCDL Day Three](#)

set description:
 dc:
 description: Pictures taken during JCDL Day Three

[JCDL Day Two](#)

Figure 6.5 Hyperlinked response from ListSets

6.6.3 Validation Procedure

The OAI protocol is a request/response protocol that works as a layer over HTTP, with responses in XML. The Repository Explorer performs validation at multiple levels in an attempt to detect the widest range of possible errors. Figure 6.6 depicts the flow of response data received from a server during the validation process.

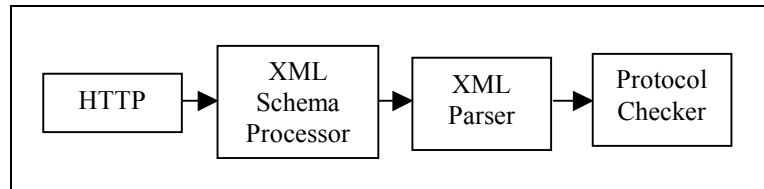


Figure 6.6 Flow of data during validation/testing process

Each step of this validation procedure performs incremental checking as described below:

1. When submitting an HTTP request, HTTP errors need to be detected and handled. The OAI protocol requires that explicitly illegal requests generate errors as HTTP status codes, and these are checked for.
2. Once the request is issued and the response is successful, the returned XML needs to be checked for validity. Since all XML responses are specified in the protocol specification using the XML Schema Language (Fallside, 2001), this part of the validation is accomplished using an XML Schema processor, which attempts to match the schema with the generated XML data stream. Many structural and encoding errors in the XML are detected in this phase of testing.
3. Unfortunately, schema processing does not always work flawlessly because of its many external dependencies, e.g., schemata are downloaded from the WWW as required. As a redundant mechanism, XML errors also are detected during the parsing and tree generation phase.
4. Lastly, the tree representation is checked for semantic correctness. For example, where controlled vocabularies are used but not encoded into the XML Schema, these are checked at this stage (e.g., the standard list of metadata formats used by the OAI).

Once checks are performed, if the software is being used in manual mode, a new interface is generated to display the data received from the server and present the user with additional options to perform further operations.

Validation may be performed using either the reference XML Schema validation toolkit available from the W3C or the Xerces XML parser (Apache Software Foundation, 2002). Local copies of both are installed to avoid further communication delays and dependencies on remote resources. In addition, the schema files for each version of the OAI protocol are installed locally to avoid the validation process being dependent on the stability of the OAI's central website. This requires that all XML responses are pre-processed to substitute OAI namespace-to-schema mappings so that they point to the local versions of the schema files.

All versions of the OAI protocol are supported by the Repository Explorer (v1.0, v1.1, and v2.0). In order to determine which version to test for, the Repository Explorer issues a transparent **Identify** request before any other requests. The protocol version is then fixed to what **Identify** reports in its *protocolVersion* field.

6.6.4 Options

The user interface provides options to change the basic appearance in terms of color scheme and language. Color schemes may be specified as CGI parameters on the command-line. The following are the default values for the color scheme, but these can be changed to improve the contrast if necessary:

```
bgcolor=aaffaa&headercolor=00cc00&blockcolor=ffcccc
```

In terms of language, there is currently user interface support for English, Chinese, Spanish, German, Korean, French, and Portuguese. Except for French and Portuguese, which were the result of machine translation, colleagues at Virginia Tech created translation tables for the other languages by hand. The language can be changed easily from the main user interface by choosing an option from the drop-down box on every page, as illustrated in Figure 6.7.

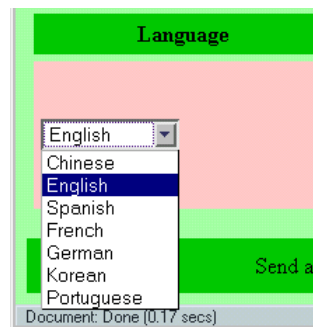


Figure 6.7 Language selection in Repository Explorer

Language selection also requires the use of an appropriate character encoding in the Web browser. Use of the Chinese interface requires installation of additional fonts while use of the Korean interface requires the specification of a Unicode encoding. All other languages work with the default “Western” encoding used in Netscape Navigator and Internet Explorer.

6.6.5 Automatic Testing

While the Repository Explorer is primarily used to help with OAI interface development, it also is used to browse through repositories for demonstrations of the OAI protocol and to show off particular collections. To make this simpler, a list of approved archives is maintained and provided as suggested starting points on the front page of the Repository Explorer. In order to maintain this list automatically, the Repository Explorer can accept

a baseURL, perform a few tests on it, and then add it to the front-page list if it passes all the tests.

For OAI-PMH v2.0 repositories, the tests submitted to the baseURL include:

- Tests for all verbs with the least number of parameters specified.
- Tests for date ranges for **ListIdentifiers** and **ListRecords**.
- Tests for correct handling of illegal verbs, parameters, parameter values, and parameter combinations.
- Tests for handling of *resumptionTokens* for **ListSets**, **ListIdentifiers**, and **ListRecords**.
- Tests for the ability to respond to parameters from previous responses, e.g., responding to **ListMetadataFormats** parameterized with an *identifier* chosen from a prior **ListIdentifiers** response.
- Tests for correlation between the *setSpec* reported in an arbitrary record header and the contents of that set, as reported by **ListIdentifiers** parameterized by *set=setSpec*.
- Tests for handling of all datestamp granularities that the archive reports support for in its **Identify** response.

These tests are synchronized with tests carried out at the OAI website (OAI, 2002) so that if a repository passes the Repository Explorer tests, it ought to be able to successfully register as an official OAI data provider.

6.6.6 Component Explorer

In order to generalize the Repository Explorer to handle ODL requests and responses, the Component Explorer was built as a specialized instance of the Repository Explorer. The prime difference is its ability to correctly identify and validate responses from ODL components. In order to determine the protocol supported, the Component Explorer looks at both the *protocolVersion* and XOAI description in the **Identify** response. If the component is an OAI archive, the Component Explorer behaves just like the Repository Explorer. Otherwise, a different set of schemata is used for XOAI validation.

Automatic testing was modified to support the inclusion of components that understand both OAI-PMH and XOAI-PMH. Thus, users may submit baseURLs for components that conform to OAI-PMH v1.1, OAI-PMH v2.0, or XOAI-PMH v1.0. When listing the components on the front page of the Component Explorer, the protocol information is automatically extracted and prepended to the component names. This is shown in Figure 6.8.

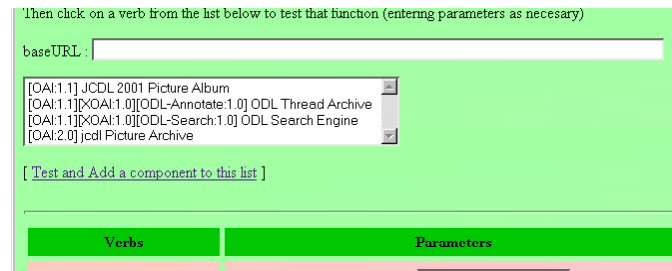


Figure 6.8 Front page listing of components in Component Explorer

Automatic testing of ODL components is not as exhaustive as is the case for OAI repositories. Basic tests are performed to ensure that the interface works, and the responses validate according to the XOAI schemata, for the administrative service requests: **Identify**, **ListSets** and **ListMetadataFormats**. However, the semantics of data transfer requests such as **GetRecord** and **ListRecords** differ greatly from one ODL protocol to another so the correctness of these cannot easily be tested with a test suite common to multiple ODL protocols. Future work may investigate solutions to this problem (as discussed later).

6.6.7 Feedback

Over the course of implementing different versions of the OAI protocol, most if not all implementers used the Repository Explorer to test their implementations. Their positive feedback supported the original motivation that a compliance test can greatly ease the process of implementation.

In addition, the design and implementation of the test software influenced the development of the OAI protocol in the following ways:

- *protocolVersion* was incorporated into the **Identify** response from an early stage in order to differentiate among different versions of the protocol.
- The transparent schema validation performed by the Repository Explorer resulted in many errors in the schemata being discovered and fixed at an early stage in the protocol development process.
- Flow control mechanisms were incorporated into an early version of the OAI protocol largely to avoid overloading of data providers, but also to ensure smaller responses that may easily be tested by tools like the Repository Explorer.

Chapter 7

ANALYSIS AND EVALUATION

7.1 INTRODUCTION

From the previously discussed case studies it is evident that information systems can be built using ODL components and by following the approach of building systems as networks of interconnected extended-OAI nodes. Measurements, analysis, and tests were carried out under controlled circumstances and in collaboration with internal and external partners in order to illustrate that the ODL approach is simple, effective, and efficient. These are discussed in this chapter.

7.2 UNDERSTANDABILITY AND SIMPLICITY

A study was performed with a group of students to gauge their level of understanding of OAI and ODL components and their ability to complete a component composition exercise satisfactorily.

The objective was to install the following components and link them together to form a simple digital library:

- XMLFile: a simple file-based OAI archive
- Harvester: an OAI/ODL harvester
- IRDB: an ODL search engine
- IRDB-ui: a simple user interface for IRDB

Figure 7.1 illustrates the architecture of the system that was built.



Figure 7.1 Architecture of simple componentized digital library

7.2.1 Methodology

Two sections of students enrolled in the CS5604 class (Information Storage and Retrieval) at Virginia Tech in Fall 2002 were given presentations on OAI and ODL,

lasting approximately 1 hour and 15 minutes for each class. Thereafter, during a lab section also lasting 1 hour and 15 minutes for each class, each student completed an exercise to install, configure, test, and connect together the above ODL/OAI components.

Students did the exercise on a Unix system where a CGI-capable Web server was already installed and properly configured. A variety of text editors were made available to accommodate different preferences among students.

Students performed the following tasks, in order, for the listed components:

- XMLFile
 - Install XMLFile file-based Open Archive.
 - Test the XMLFile component using different testing approaches (as discussed in the previous chapter).
 - Add more data to the component and test for correct dissemination of updates.
- Harvester
 - Install a harvester.
 - Configure it to harvest data from the XMLFile archive.
 - Perform a preliminary harvest of the data to test it.
- IRDB
 - Install the IRDB search engine.
 - Configure it to harvest data from the XMLFile archive.
 - Harvest the data and perform a series of tests with various queries.
- IRDB User Interface
 - Install the IRDB user interface.
 - Configure it to use the IRDB component already installed.
 - Test the interface using a web browser and various queries.

Students then were asked to fill out a questionnaire to evaluate the experience of building this system from components. 56 students completed the survey completely. Two completed only one page of the questionnaire and so were excluded from the analysis.

7.2.2 Results

Table 7.1 provides a summary of the responses for the background information section of the questionnaire.

Question	Responses
Major/Department	CS : 56
Program level and year	Undergrad – 3 rd year : 1 Masters – 1 st year : 19 Masters – 2 nd year : 28 Masters – 3 rd year : 1 PhD – 1 st year : 5 Visiting Professors : 2
Have you used Unix or a Unix-like operating system before (e.g., FreeBSD, Solaris, Linux) – at least to the extent of logging in and editing text files?	Yes : 52 No : 4
Have you used a component-based development environment before (e.g., Delphi, C++/JBuilder, Visual C++/J++, JavaBeans)?	Yes : 50 No : 6
Have you installed a CGI web-server script/application before (e.g., website guestbook, website counter)?	Yes : 25 No : 31
Have you developed a CGI web-server script/application before?	Yes : 22 No : 34
Do you know how Web Services works (e.g., involving SOAP, WSDL, UDDI)?	Yes : 27 No : 29
Have you done any Web Services development?	Yes : 22 No : 34

Table 7.1 Results to background information questions on survey

Thereafter the survey contained a number of questions related to the exercise to assess the students' reactions to the exercise and the underlying tools and techniques. Most questions in this section were framed using a 5-point Likert scale. The last question asked for freeform "comments, concerns or suggestions". Table 7.2 summarizes the responses to all but the last question.

Question	Response				
	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
I understand the basic concepts of the OAI Protocol for Metadata Harvesting.	9	38	9		
I understand the basic concepts of ODL components.	6	36	14		
The instructions for the exercise were understandable.	35	20	1		
Installing the components was simple.	36	18	2		
Configuring the components was simple.	33	21	2		
Connecting together the Harvester and XMLFile components was simple.	28	25	3		
Connecting together the IRDB and XMLFile components was simple.	26	25	5		
Carrying out this exercise has improved my understanding of the concepts underlying OAI and ODL.	13	25	12	6	
I would consider using ODL and OAI components if I need to build a system with requirements similar to the exercise.	12	33	10	1	

Table 7.2 Responses to questions on exercise

Responses to the last question were varied and some are discussed in the following section.

7.2.3 Discussion

The different backgrounds of participants evident from the responses to demographic questions makes it difficult to analyze these results without taking into account all of the interaction effects that result from past experience. Since OAI and ODL utilize various different Web technologies, it is non-trivial to enumerate all of the pre-requisites and determine their independent effects. It may be possible to construct an experimental model to minimize the interaction effects, but this will require finding unique participants, each with a very particular background and training. This may prove difficult because of the cutting-edge nature of technology used by OAI and ODL. Taking these difficulties into account, any analysis of such an experiment cannot easily determine general trends.

Nine respondents who indicated that they did not know how Web Services worked answered affirmative when asked if they had done Web Services-related development. This may be because they interpreted the question as referring to Web-related services other than SOAP (Box, et al., 2000), WSDL (Ogbuji, 2000), and UDDI (Ariba Inc., 2000), or because they had done development work without understanding the underlying

standards and information model of Web Services. Either of these is consistent with the vague understanding that many people have of Web Services.

Judging from the responses to the first two questions in Table 7.2, most participants appear to have grasped the basic concepts related to OAI and ODL. The fact that some participants were unsure indicates that an hour and 15 minutes may not be enough for a person building a digital library to learn enough about OAI and ODL. This raises the question of just how much training a person needs before being able to effectively use OAI and ODL technology. Also, more of the participants were able to understand OAI than ODL; this is expected since ODL builds on OAI. Further, in a separate discussion reported to the experimenter, one student explained to the course instructor that his learning style was not hands-on, and more oral explanation along the way when doing the experiment would have made it clearer what was going on.

Most participants agreed (or strongly agreed) that the instructions were understandable. The instructions were very detailed so that even if participants did not understand one section of the exercise, they were still able to complete the rest of the steps.

Installation and configuration of individual components as well as interconnecting different components was deemed to be simple. As these are two basic concepts underlying ODL (that all services can be independent components and that systems are built by interconnecting service components), it supports the hypothesis of this experiment that ODL is simple to understand and adopt.

There was not much agreement about the ability of the exercise to improve the participants' understanding of OAI and ODL. This can be attributed to the sheer volume of new concepts covered during the presentation and exercise. Given that approximately half of the participants had never created a CGI-based web application before, the learning curve was quite steep. In practice, those who adopt OAI and ODL technology are usually digital library practitioners who already have experience with the construction of dynamic Web-based information systems.

In spite of all these factors, two-thirds of the participants indicated an interest in using similar components if they have a need for such services. Thus, even without a thorough understanding of the technology and other available options, the simple and reusable nature of the components seemed to appeal to participants.

Thirteen participants provided optional feedback in the last question, and these ranged from positive to somewhat skeptical. Eight of the reactions were positive, including the following comments:

- "I think the idea is very good and the approaches to build digital libraries is easy."
- "They provide a way to get up and running very quickly with a Web application."

Some participants were not sure about the workflow as indicated by the comment:

- "We have high level idea but detailed explanation will be great."

One comment in reference to the questions on simplicity of installation and configuration included:

- “I don’t know; custom config might not be simple.”

This summarizes the notion that components should be simple enough to bootstrap a development process but still powerful enough to support a wide range of functionality. In particular, the above comment refers to the XMLFile component that is simple to install and use in its default configuration, but can be non-trivial to configure if the records are not already OAI-compatible. In such a situation, XSL transformations can be used to translate the records into acceptable formats. However, irrespective of the complexity of configuration for a particular instance, the OAI/ODL interface to such components always is the same.

Some questions raised during the lab sessions revealed very important issues that need to be addressed in future development of ODL or related standards:

- Confusion over baseURLs
 - o Some participants were confused regarding which baseURL to use in which instance. Since all URLs were similar, it was not obvious – this will happen in practice with any system built on OAI, ODL, or Web Services technology.
 - o Entering URLs by hand resulted in many typographical errors. Ideally, such links must be made using a high-level user interface that masks complex details like URLs from the developers.
 - o The user interface was sometimes connected to the wrong component. While it is possible for a user interface to **Identify** the service component before using it, this will be inefficient. As an alternative, user interfaces can themselves be components, with associated sanity tests applied during configuration.
- Failures during harvesting
 - o Harvesting will fail if the baseURL is incorrect, but there are no obvious graceful recovery techniques. The components used in the experiment assume a catastrophic error and stop harvesting from the questionable archive pending user intervention. Better algorithms can be devised to implement exponential back-off and/or to trigger notification of the appropriate systems administrator.

7.2.4 Conclusions

This exercise has demonstrated that:

- It is possible for people with little experience to quickly learn the basic concepts related to OAI and ODL.

- OAI/ODL component installation and configuration and the composition of components are relatively simple and understandable processes.

7.3 REUSABILITY

To demonstrate reusability of ODL components and protocols, collaborators at Virginia Tech and other institutions were encouraged to use the components in their projects. Some of these efforts are discussed below.

7.3.1 Case study: AmericanSouth.org

AmericanSouth.org (Halbert, 2002) is a collaborative project led by Emory University to build a central portal for scholarly resources related to the history and culture of the American South. The project was initiated as a proof-of-concept test of the metadata harvesting methodology promoted by the OAI. Thus, in order to obtain data from remote data sources, the project relies mainly on the OAI-PMH.

The requirements for a central user portal include common services such as searching and browsing. AmericanSouth.org used ODL components to assist in building a prototype of such a system. The DBUnion, IRDB, and DBBrowse components were used in addition to XMLFile and other custom-written OAI data provider interfaces. Many questions about protocol syntax and component logic were raised and answered during the prototyping phase, suggesting that more documentation is needed. Alternatively, pre-configured networks of components can be assembled to avoid configuration of individual components. Both of these approaches are being investigated in the DL-in-a-Box project (Luo, 2002).

The production system for AmericanSouth.org still uses multiple instantiations of XMLFile but the ODL components have been replaced with the ARC search engine (Liu, et al., 2001) largely because of concerns over execution speed of the IRDB search engine component.

7.3.2 Case study: CITIDEL

CITIDEL – the Computing and Information Technology Interactive Digital Education Library (Fox, et al., 2002a) – is the computing segment of NSF’s NSDL – the National Science, Technology, Engineering and Mathematics Digital Library (Lagoze, et al., 2002a). CITIDEL aims to build a user portal to provide access to computing-related resources garnered from various sources using metadata harvesting wherever possible. This user portal is intended to support typical resource discovery services, such as searching and category-based browsing, as well as tools specific to composing educational resources, such as lesson plan editors.

From the initial stages, CITIDEL was envisioned as a componentized system, with an architecture that evolves as the requirements are refined. The initial system was designed to include multiple sources of disparate metadata and multiple services that operate over this data, where each data source and service is independent. This is depicted in Figure 7.2.

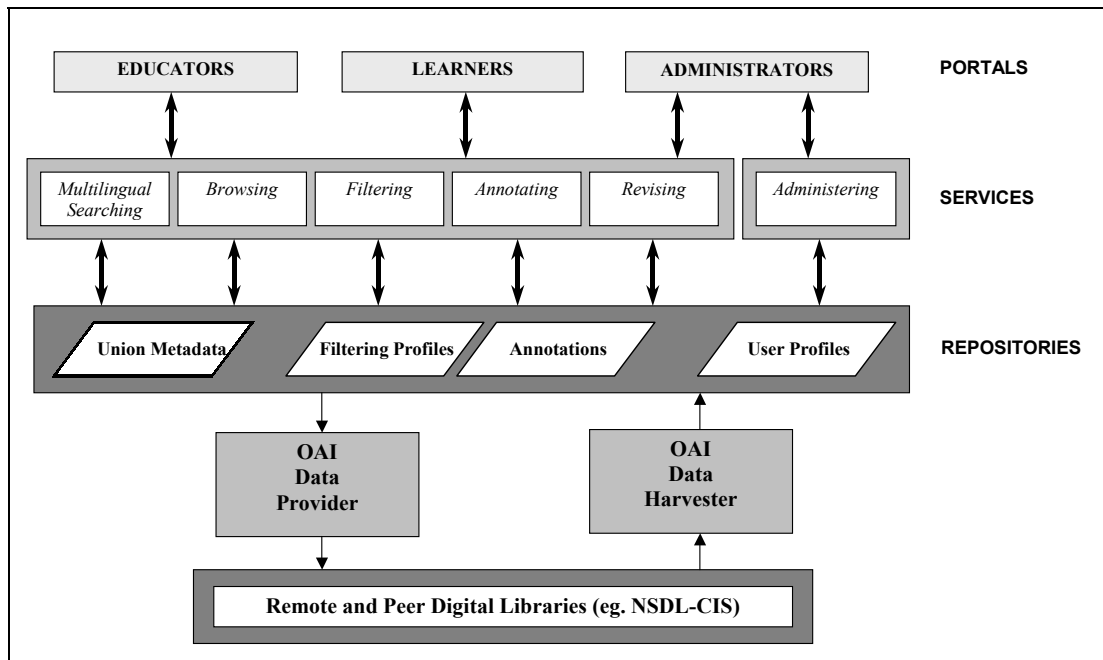


Figure 7.2 Original architecture of CITIDEL, showing metadata layer distinct from service layer

CITIDEL uses components from various sources. In terms of ODL, this includes the IRDB and Thread components to implement simple searching and threaded annotations, respectively. The IRDB component was modified to make more efficient use of the underlying database.

CITIDEL is closely related to the JERIC and CSTC projects discussed in the previous chapter – both provide avenues for submission of resources, the metadata for which is ultimately harvested by CITIDEL and exposed to users through its user interface.

7.3.3 Case study: BICTEL/e

The BICTEL/e project, led by the Universite Catholique de Louvain, is building a distributed digital library of dissertations and e-prints within the nine French-speaking universities in Belgium.

The project adopted use of the following OAI and ODL components:

- ETD-db OAI extension
- DBUnion archive merger component
- DBBrowse collection indexer and browser

Each university will maintain independent collections for dissertations and e-prints, merged together with the DBUnion component. User-level services will be provided via the DBBrowse component and other non-ODL service components (such as a search engine). The central site has a similar system in place, differing only in that metadata is harvested from remote sites rather than from local collections.

This project is still in the developmental stages – the first participating site and the central site have been set up. Additional sites are in the testing phase.

7.4 EXTENSIBILITY

7.4.1 Sub-classing

Some component implementations were created by sub-classing existing components. All of the component modules were written in object-oriented Perl, which allows for single inheritance, so this was exploited when possible. Since the DBRate and DBReview components also store the original transaction records submitted to them, they were derived from the Box component. In each case, some of the methods were overridden to provide the necessary additional functionality.

7.4.2 Layering

7.4.2.1 VIDI

The VIDI project (Wang, 2002) developed a standard interface, as an extension of the OAI protocol, to connect visualization systems to digital libraries. A prototype of the VIDI reference implementation links into the search engine of the ETD Union Catalog to obtain search results. The search engine used in the ETD Union Catalog understands the ODL-Search protocol, which also is based on the OAI protocol. Thus, additional services are provided as a layer over an ODL component, without any reciprocal awareness necessary in the ODL system.

7.4.2.2 MAIDL

MAIDL, Mobile Agents In Digital Libraries (Nava Muñoz, 2002), is a federated search system connecting together heterogeneous web-accessible digital libraries. The project uses the “odlsearch1” syntax, as specified in the ODL-Search protocol, in order to submit queries to its search system. Further communication among the mobile agents and data providers transparently utilize the XOAI-PMH protocol.

7.4.2.3 VTLS Union Catalog

VTLS operates a production version of the ETD Union Catalog for NDLTD using an instance of their Virtua library management software (VTLS, 2002). Thesis and dissertation metadata is gathered from various institutions in multiple formats. VTLS also harvests metadata directly from the DBUnion component of the ODL-based Union Catalog (discussed as a case study in Chapter 5). Thus, Virtua works as a service provider layer over the ODL component.

7.5 PERFORMANCE

Performance evaluation was done in terms of the following aspects:

- Communications and protocol overhead incurred by OAI/XOAI protocols
- Execution speed of nested components
- Execution speed optimizations
- Load analysis
- User interface response
- Storage required for components and the effects of duplication
- Consistency among various copies of data stored on different nodes
- Harvesting algorithms and their efficiencies in terms of speed and network utilization

7.5.1 Communications and Protocol Overhead

Measurements of response times were taken at various layers within a single ODL component in order to determine the execution time overhead associated with network communications and the ODL protocols.

7.5.1.1 Methodology

The IRDB search engine component was used for this test because search operations take a non-trivial (and therefore measurable) amount of time and the pre-packaged component includes a direct interface to the search engine that allows bypassing of the ODL protocol layer.

The tests were conducted on a dedicated 2Ghz Pentium 4 PC with 1GB of RAM running the pre-installed version of Red Hat Linux v7.3. The Web server used was Apache v1.3.23 with the default configuration, supplemented only by changes to allow for execution of CGI scripts. Data for all components was stored in a MySQL v3.23.49 database.

For test data, a mirror of the ETD Union Archive was created and this then was harvested and indexed by an instance of the IRDB component. 7163 items were contained in this collection, each with metadata in the Dublin Core format.

The test was to execute a search for a given query. Three queries were used: “computer science testing”, “machine learning”, and “experiments”. At most the first 1000 results were requested in each case. Each query was executed 100 times by a script to minimize the effect of the script on the overall performance. The first run of each experiment was discarded to minimize disk access penalties, and an average of the next 5 runs was taken in each case.

For this experiment, only the identifiers were requested. Obtaining the full records would have required communication with the source archive – this was measured in a later experiment.

Six runs were made for each query:

1. Executing *lynx* to submit a **ListIdentifiers** query through the Web server interface.
2. Executing *wget* to submit a **ListIdentifiers** query through the Web server interface.
3. Using custom-written HTTP socket code to submit a **ListIdentifiers** query through the Web server interface.
4. Executing the search script directly from the command-line, thereby bypassing the Web server.
5. Executing *testsearch.pl* to bypass both the Web server and the ODL layer.
6. Using direct API calls to the IR engine, without spawning a copy of *testsearch.pl* in each iteration.

Figure 7.3 is a pictorial representation of the layers at which the tests were conducted. The numbers correspond to the tests listed above. Tests 1, 2 and 3 used communication with the component indirectly through the Web server. Test 4 used communication with the ODL-Search interface. Test 5 used communication with the information retrieval engine used by IRDB. Test 6 is indicated as being within the component since it used a modified version of component’s interface script for measurement.

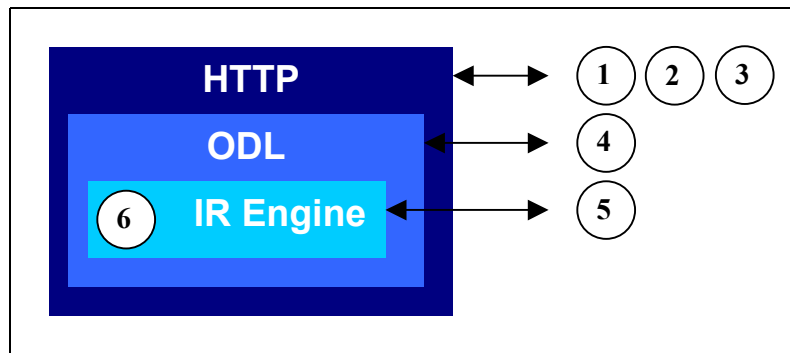


Figure 7.3 Testable interfaces for IRDB component

The time was measured as the “wall-clock time” reported by the *bash* utility program *time* from the time a run started to the time it ended. The script that ran the experiment controlled the number of iterations (100, in this case) and executed the appropriate code in each of the 6 cases above. In each case, the output was completely collected and then immediately discarded – thus, each iteration contributed the complete time between submitting a request and obtaining the last byte of the associated response, hereafter referred to as the execution time.

7.5.1.2 Results

The average times (in seconds) obtained for each test are listed below in Table 5.5.

Query	Test-1	Test-2	Test-3	Test-4	Test-5	Test-6
a. "computer science testing"	54.82	50.30	49.26	48.59	48.26	39.70
b. "machine learning"	31.57	27.85	27.22	26.64	26.43	15.81
c. "experiments"	47.52	43.26	42.32	41.97	41.31	32.78

Table 7.3 Execution times for request submitted to different layers of ODL-IRDB

In order to understand the reasons for differing times, additional runs were conducted without discarding the output in order to determine the number of matches for each query. These are indicated in Table 7.4.

Query	Matches
a. "computer science testing"	2478
b. "machine learning"	463
c. "experiments"	1020

Table 7.4 Number of matches for each query

7.5.1.3 Discussion

From Table 7.3 we notice that the execution time increases as more layers are introduced into the component. This increase is not always a large proportion of the total time, but the difference between Test-1 and Test-6 is significant. The time differences between pairs of consecutive tests is indicated in Table 7.5.

Query	Test1-2	Test2-3	Test3-4	Test4-5	Test5-6
a. "computer science testing"	4.52	1.04	0.67	0.33	8.57
b. "machine learning"	3.72	0.63	0.58	0.22	10.62
c. "experiments"	4.26	0.94	0.35	0.66	8.53

Table 7.5 Time differences between pairs of consecutive tests

Test-1, Test-2, and Test-3 illustrate the differences in times due to the use of different HTTP clients. In Test-1, the fully-featured text-mode Web browser *lynx* was used. In Test-2, *wget* was used instead, and the performance improved because *wget* is a smaller application that just downloads files. Test-3 avoided the overhead of spawning an external client application altogether by using custom-written network socket routines to connect to the server and retrieve responses to requests. The differences are only slight but there is a consistent decrease for all queries.

The difference between Test-3 and Test-4 is due to the effect of requests and responses passing through the HTTP client and the Web server. While no processes were spawned at the client side in Test-3, a process was still spawned by the Web server to handle each request at the back-end. This script was run directly in Test-4, so the difference in time is

due solely to the request being routed through the Web server. This difference is small, so it suggests that the Web server does not itself contribute much to the total execution time.

The difference between Test-4 and Test-5 is due to the ODL-Search software layer that handles the marshalling and unmarshalling of CGI parameters and the generation of XML responses from the raw list of identifiers returned by the IR engine. This difference is also small, indicating that the additional work done by the ODL layer does not contribute much to the total time of execution.

The difference between Test-5 and Test-6 is due to the spawning of a new process each time the IRDB component is used. This difference is substantial and indicates that process startup is a major component of the total execution time.

The number of matches for each query, as shown in Table 7.4, indicate that the execution times for the chosen queries are not determined solely by the number of terms but also by the popularity of the terms. In this experiment and others where the maximum number of results requested was 1000, Query-b had understandably faster execution times because fewer results were obtained from the inverted files and subsequently formatted and returned to the client. In experiments where the number of results requested was less than 463 (the total number of matches for Query-b), the number of terms can play a more significant role in execution time differences.

7.5.1.4 Conclusions

The execution times for the IRDB component (as representative of ODL components in general) were much higher than the execution times for direct API calls. However, this difference in execution time is due largely to the spawning of new processes for each request. The ODL layer and the Web server contribute only a small amount to the total increase in execution time.

7.5.2 Execution Speed: Nested Components

In order to avoid duplication of metadata entries, some of the ODL components (such as IRDB and DBBrowse) do not store redundant copies – instead, every time a record is needed, it is fetched from the source archive. This procedure is hereafter referred to as a nested request.

The following test measured the comparative execution times for **ListIdentifiers** (single) and **ListRecords** (nested) requests for the IRDB component.

7.5.2.1 Methodology

The IRDB search engine component was used for this test. The test was performed in the same experimental environment as for the *Communications and Protocol Overhead* experiment.

The test was to execute a search for a given query. Three queries were used: “computer science testing”, “machine learning”, and “experiments”. At most the first 10 results

were requested in each case. Each query was executed 10 times by a script. The first run of each experiment was discarded and an average of the next 5 runs was taken in each case.

The ODL interface was invoked by direct execution, thus bypassing the Web server (i.e., the “Test-4” procedure in the previous experiment was employed here).

Two runs were made for each query:

1. Submitting **ListIdentifiers** to get a list of matching identifiers.
2. Submitting **ListRecords** to get a list of matching records.

7.5.2.2 Results

The average times (in seconds) obtained for each test are listed below in Table 7.6.

Query	1. ListIdentifiers	2. ListRecords	(ListRecords-ListIdentifiers)
a. “computer science testing”	1.57	16.36	14.79
b. “machine learning”	1.47	16.22	14.75
c. “experiments”	1.44	16.21	14.77

Table 7.6 Response times for ListIdentifiers vs. ListRecords

7.5.2.3 Discussion

A **ListRecords** query issued to the ODL-Search component requires a single **ListRecords** request as well as one **GetRecord** request, issued to the source archive, for each record in the result set. The records returned from these GetRecord requests are merged to form the response to **ListRecords**. Thus, in general,

Let the number of records in the result set = $n = 10$

Let the execution time = e

Let the time taken for component startup and processing = s

Let the process spawning time = p

Now, **ListIdentifiers** needs 1 request (process is spawned by the test script), but

ListRecords needs $1 + n$ requests.

Then, for **ListIdentifiers**: $e(li) = s + p$

And for **ListRecords**: $e(lr) = s + p + n p$

Since s and p are approximately equal for both requests, we can simplify this to:

$$p = (e(li) - e(lr)) / 10$$

Substituting values from Table 7.6, we find that $p=1.479$, $p=1.475$, and $p=1.477$ for queries a, b, and c respectively. This confirms that startup time is approximately constant and that the model for requests and responses is accurate, where **ListRecords** takes an additional amount of time that is proportional to the number of records in the result set.

Resubstituting into the equation for $e(li)$, we find $s=0.09$, $s=0.01$, and $s=0.04$ for queries a, b, and c respectively. The slight differences in time are due to the differing numbers of matching records, as indicated in Table 7.4.

7.5.2.4 Conclusions

This experiment confirmed that:

- The time taken for process startup is a large portion of the execution time.
- The time taken for nested requests depends on the number of items in the result set.

Various techniques were subsequently investigated to optimize components to minimize the effect of each of these factors. These are discussed in the next section.

7.5.3 Execution Speed Optimizations

There is an obvious performance penalty due to layering and use of a Web server with back-end scripts. Various tools have been developed to address this. As a representative of these tools and techniques, the SpeedyCGI package (SpeedyCGI, 2002) was tested and compared against the case where no optimizations are used. SpeedyCGI is a tool that speeds up access to Perl scripts without modification of the script or the Web server. Instead of running Perl with each invocation of a script, the script is run by a relatively small SpeedCGI front-end program that connects to a memory-resident Perl back-end, creating the back-end process if necessary. Thus, the process startup time is determined by the execution speed of the front-end script rather than the Perl interpreter.

The objectives of this study were:

- To calculate the response times for single and nested requests, both with and without using SpeedyCGI.
- To compare SpeedyCGI usage with the fastest approach thus far, that of directly utilizing a programming API.

7.5.3.1 Methodology

The IRDB search engine component was used for this test. The test was performed in the same experimental environment as for the *Communications and Protocol Overhead* experiment.

The test was to execute a search for a given query. Three queries were used: “computer science testing”, “machine learning”, and “experiments”. The first run of each experiment was discarded and an average of the next 5 runs was taken in each case.

For the first part of the experiment, all requests were submitted by executing *wget*, thus involving the Web server and the ODL interface in generation of the response. At most the first 10 results were requested in each case. Each query was executed 10 times by a script. Four runs were made for each query as follows:

1. By submitting **ListIdentifiers** (LI).
2. By submitting **ListRecords** (LR).
3. By submitting **ListIdentifiers**, where the components use SpeedyCGI (LIS).
4. By submitting **ListRecords**, where the components use SpeedyCGI (LRS).

For the second part of the experiment, at most 1000 results were requested and each query was executed 100 times. The requests were submitted by executing *wget*, thus involving the Web server and the ODL interface in generation of the response. A single **ListIdentifiers** run was conducted for each query, and this was contrasted with the data obtained during the direct API measurements taken in the *Communications and Protocol Overhead* experiment.

7.5.3.2 Results

Table 7.7 displays the comparisons from the first part of the experiment – using SpeedyCGI and not, for both **ListIdentifiers** and **ListRecords** requests submitted to IRDB.

Query	Regular CGI		SpeedyCGI	
	1. LI	2. LR	3. LIS	4. LRS
a. "computer science testing"	1.67	16.50	0.44	2.22
b. "machine learning"	1.57	16.34	0.33	2.04
c. "experiments"	1.55	16.35	0.31	2.06

Table 7.7 Regular CGI vs. SpeedyCGI speed comparisons

Table 7.8 displays the comparisons from the second part of the experiment – using SpeedyCGI and comparing this to the previous measurements for the case with direct API use.

Query	SpeedyCGI	API
a. "computer science testing"	40.27	39.70
b. "machine learning"	16.61	15.81
c. "experiments"	32.39	32.78

Table 7.8 SpeedyCGI vs. direct API speed comparisons

7.5.3.3 Discussion

Results from the first part of the experiment indicate that there is a significant improvement in execution speeds for both **ListIdentifiers** (single requests) and **ListRecords** (nested requests) when SpeedyCGI is used. This is largely due to the elimination of the need to spawn new processes to handle each request to the Web server.

The second set of results indicate that there is very little difference in execution times between using direct API calls and using a fully layered IRDB component when SpeedyCGI is used.

7.5.3.4 Alternative Optimizations

In addition to SpeedyCGI, many other alternatives were considered to optimize the operation of ODL components. Some of these are discussed below.

Direct Database Access

If two components such as IRDB and DBUnion are collocated on the same machine, it may be possible for one to access the database of the other directly. This was tested in the initial prototypes. However, since it is not portable and does not conform to the design goals of componentized systems in general, it was not pursued further.

Union Catalog Specialization

Since the most common request from IRDB is **GetRecord** issued to the DBUnion component (or CSTC OAI interface), its communication can be optimized. Without deviating from the model of the ODL network, the DBUnion was reimplemented in C++ with only a **GetRecord** service request handler. Then, all other components used this special interface to get lists of records. This implementation was noticeably faster, and demonstrated that the programming language of choice can influence the speed of component interaction.

GetRecords

Requesting multiple records together in a batch can drastically reduce the number of network requests. This can be achieved by extensions to the OAI protocols using sets to indicate lists of elements or by using a new service request. Before investigating these approaches from an ODL perspective, this issue was forwarded to the OAI Technical Committee for possible incorporation into version 2.0 of the OAI-PMH. There was much discussion but the consensus was that it would not assist in batch metadata harvesting operations. A “GetRecords” batching feature can still be incorporated into a future version of XOAI-PMH or, more specifically, the ODL-Union protocol.

mod_perl

A popular technique among Perl programmers who desire faster speeds is to use the mod_perl Apache module, which embeds a Perl interpreter into the server itself. This usually works well but is limited to Perl and creates a much larger memory footprint for

the server. There is a similar module for PHP, but it has the same drawbacks. This solution was not tested because of the inherent language limitations, coupled with the necessary changes to the Web server.

Direct Code Access

With collocation of components, it may be possible for one component to call the functions and procedures of the other directly. This requires some innovative planning in laying out the components and linking them together so that they invoke local copies if possible but otherwise default to HTTP. This was tested and the performance improved; it was similar to the “Test-4” case of the *Component and Protocol Overhead* experiment. However, this technique results in a high degree of coupling, which violates the independence criterion for component systems.

Non-forking Web Server

Besides the popular Apache, there are various other Web servers available commercially and a handful of non-commercial experimental ones, e.g., Boa (Philips, et al., 2002) and thttpd (ACME Labs, 2002). Among other, Boa and thttpd claim to be faster because they are lightweight and non-forking (some servers use threads but others poll network connections within a single thread for maximum speed). Since process creation takes a significant amount of time, this claim sounds tempting. However, upon investigation, none of these servers offer nearly the same suite of features supported by Apache so they will not make good general-purpose servers. Also, none of them will speed up spawning of CGI-type applications. All things considered, this may not significantly improve the performance of ODL components.

Special-purpose Servers

Some projects for which speed is absolutely essential will embed a Web server into a script. This means that the script is persistent and there is no switching from Web server to script and vice versa. The problem with this approach is that it is very specific and means that every component has to have a persistent server in memory. With many components in a typical ODL network, this is not likely to be feasible, especially for small DLs. Also, it significantly raises the bar on programming requirements. It was not pursued.

Persistent Components

Even if a component does not include a Web server, it may be possible for the component to stay in memory and be glued into the Web server whenever necessary by a much smaller script. This has the same disadvantages as the above approach and requires the design of a special protocol for communication between the Web server and the component. Instead of custom-building such a solution for each component, the SpeedyCGI toolkit provides comparable services without requiring any code or Web server modification.

FastCGI

FastCGI (Brown, 1996) is an add-on kit that provides persistent script capabilities to a Web server, independently of the programming language. Scripts need to be modified slightly by encapsulating them in a simple loop but this is relatively minor and for some components it was possible to create both regular and FastCGI versions without much change. FastCGI provides an add-on server module that loads a script on demand and keeps it persistent, with support for dynamic reloading and dynamic load balancing. This was tested informally and had performance characteristics comparable to SpeedyCGI. There are additional security problems that need to be resolved since FastCGI enforces a higher level of security than regular scripts, but better programming discipline and security is good for component development, so this can be seen as another advantage.

JAVA Servlets

JAVA is a programming language of choice for many programmers on the Web, so there are just as many tools available as there are for languages such as Perl. JAVA servlets (Zeiger, 1999) are pieces of JAVA code that stay persistent in the Web server and thus achieve much better performance. While this is widely used, it requires the use of a JAVA byte-code interpreter and therefore does not result in substantially better performance than persistent Perl scripts.

Caching

Using caching at various levels within the experimental systems resulted in significant speed improvements. For example, in the ODL-based ETD Union Catalog, the DBBrowse component caches the results from DBUnion, thus minimizing the number of recurring requests. Secondly, the user interface caches the responses to most requests; thus speeding up the process of browsing through a list of returned items. Together, these have a noticeable effect on execution speed. One problem that manifests itself was that of stale data in a cache. It is still being investigated – there are ways to force a refresh from the Web browser to propagate to the server's scripts, but this apparently only works for Netscape browsers and works differently in each version. For more systems that are more dynamic (such as CSTC), caching is not used because changes have to propagate through the system immediately.

7.5.3.5 Conclusions

SpeedyCGI is a very simple Web server optimization module that is easily installed on Linux and other Unix and Unix-like systems. By using this module to avoid the time penalties incurred from making intermediate Web server requests, it was shown that ODL components can have comparable execution times to direct API calls.

In addition to the use of SpeedyCGI and similar Web application persistence tools (FastCGI, JAVA servlets, `mod_perl`, etc.), caching and the ability to request multiple records in a single request show promise to improve the performance of ODL components. Further investigation is needed.

7.5.4 Load Analysis

Under real-world conditions, response times can be drastically different as situations vary. The aim of this experiment was to assess the ability of a component to perform acceptably under high loads. To assess this, a server was artificially loaded and then the response times of typical requests were measured under different load conditions.

7.5.4.1 Methodology

The Box component was used for this test because it has very little component logic and therefore provide lower bounds for execution speed that are indicative of the ODL component architecture and not the component logic.

The component was installed in the same server environment used in the *Communications and Protocol Overhead* experiment. A second identical machine was used to simulate client machines by running multiple processes, each of which submitted **ListRecords** requests to the server in a continuous loop. The server was primed with 100 dummy records for this purpose.

The test was to submit **GetRecord** and **PutRecord** requests to the local server. The first run of each experiment was discarded and an average of the next 5 runs was taken in each case. The experiment was then repeated using SpeedyCGI for the Box component.

7.5.4.2 Results

Table 7.9 lists the average execution times for **GetRecord** and **PutRecord** operations under load conditions generated by 5, 10, and 50 simultaneous processes.

	5 clients	10 clients	50 clients
PutRecord	1.04	2.39	9.31
GetRecord	1.39	2.06	11.27

Table 7.9 Average execution times under different load conditions

Table 7.10 lists the average execution times for **GetRecord** and **PutRecord** operations under load conditions generated by 5, 10, and 50 simultaneous processes, when the Box component uses SpeedyCGI.

	5 clients	10 clients	50 clients
PutRecord	0.691	0.702	2.146
GetRecord	0.449	0.316	1.801

Table 7.10 Average execution times when using SpeedyCGI

7.5.4.3 Discussion

There is variability in execution time because of the non-deterministic nature of client-server synchronization and process startup. It is apparent, however, that the time taken to respond to a request increases as the load on the server increases.

Using the persistent script mechanism of SpeedyCGI results in a reduction of the execution time as compared to the no SpeedyCGI case, but there is still an increase with increasing load, as expected. Figure 7.4 shows these relationships graphically.

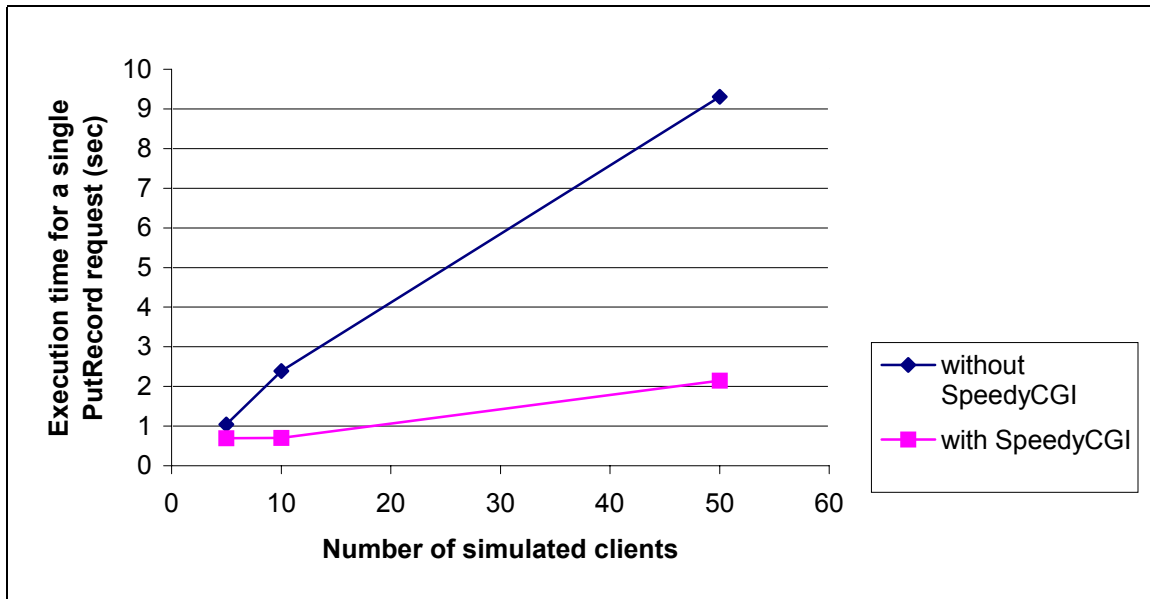


Figure 7.4 Load conditions with SpeedyCGI and without

7.5.4.4 Conclusions

A high load on the server causes an increase in execution time for component interaction. The SpeedyCGI module, as representative of persistent script tools, helps to minimize this effect. Ultimately, however, ODL components are Web applications and the only way to get better performance for a heavily loaded Web server is to use more and/or faster servers.

7.5.5 User Interface Response

While inter-component communications speeds are important to system designers, it is the speed of the user interface that matters the most to users. To test this, requests were submitted to the new CSTC interface to determine its effectiveness.

The objective was to submit requests to the new CSTC interface, simulating typical user behavior, and then measure the response time.

7.5.5.1 Methodology

The client machine was a 2Ghz Pentium 4 PC with 1GB of RAM running a pre-installed version of Red Hat Linux v7.3.

The server used was a non-dedicated 600MHz Pentium 3 with 256MB RAM running Red Hat v6.2. The Web server was Apache v1.3.12 and data for all components was stored in a mySQL v3.23.39 database. All components used the SpeedyCGI tool to remain persistent in memory.

The first test involved simulating a browse operation. The second test involved simulating the viewing of metadata for a single resource. In both instances, the test was repeated 10 times per invocation of the test script. The test script was executed 5 times and the average of these was computed.

7.5.5.2 Results

Table 7.11 displays the user interface execution times for the operations tested.

Action	t (Time taken for 10 requests)	t / 10
Browse first screen of items	9.28	0.93
Display metadata for first item	9.81	0.98

Table 7.11 Execution times for user interface actions

7.5.5.3 Discussion

The browsing operation required 1 request that was submitted to the DBBrowse component, as well as 5 nested requests sent to the DBUnion component in order to fetch the metadata, resulting in a total of 6 requests. The display operation required 1 request for the metadata, 1 for the rating, 4 for the recommendation, and 3 for the feedback mechanism – resulting in a total of 9 requests.

The time taken is not simply proportional to these request counts because different components contribute varying amounts to the total execution time. However, as indicated in Table 7.11, the total response time in both instances is <1 second for data transfer.

7.5.5.4 Conclusions

The time taken to generate user interface pages is reasonably small for the new CSTC system, running on a production server.

7.5.6 Storage

Many components store data in a database, and these databases need indices in order to be accessed efficiently. Store-and-retrieve OAI archives do not incur many penalties since there is little redundancy within a single archive. However, some ODL components

create duplicates of some or all of the data. DBBrowse and WhatsNew create indices while IRDB creates inverted files. The first two components have minimal storage requirements and do not pose any serious problems.

The search engine, on the other hand, requires space for the inverted files because the larger indices tend to support faster retrieval. This has been highly optimized to support reasonable speed of execution for the prototype, but at the expense of greater storage requirements. A more sophisticated search engine can be used to avoid these problems. Alternatively or in addition, better database technology can be used. The MG system used in the Greenstone project used inverted file compression to achieve compression of up to 96% compared to the source text (Witten, et al., 1999).

The duplication of actual metadata can make componentization prohibitively costly. However, all metadata can be stored in a central repository or union repository and accessed when needed. This is the approach that is taken in most of the experimental systems, including the new CSTC system and the ODL-based ETD Union Catalog.

Components that do not harvest data – such as DBRate and DBReview – do not incur additional storage penalties.

7.5.7 Duplication of Data

Suppose that the minimum difference between two timestamps (granularity) is g (in days). Then, the maximum difference between the timestamp and the actual time of modification of a record is g . This implies that in order to ensure data consistency, a harvester has to overlap harvesting operations by at least g . (When archives operate in different timezones, the interpretation of a timestamp differs by at most a day, necessitating an overlap of $(1+g)$ days.)

When component A harvests data from a source archive, it has to update all timestamps so that changes will propagate to any downstream components. Thus, component B that harvests from component A will deal with a new system of dates, necessitating an additional overlap of g . Compared to the source archive, there will now be a time interval overlap of $2g$ in the records being harvested. Table 7.12 illustrates this incremental duplication when 2 components are connected in a chain to a source archive and there is an overlap of 1 day. In the illustration items are added at the source, Component A harvests daily from the source, and Component B harvests daily from Component A.

Source		Component A	Component B
Item (Date)		Item (Date)	Item (Date)
Mon	Item 1 (Mon)	Item 1 (Mon)	Item 1 (Mon)
Tues	Item 2 (Tues)	Item 1 (Tues)	Item 1 (Tues)
		Item 2 (Tues)	Item 2 (Tues)
Wed	Item 3 (Wed)	Item 2 (Wed)	Item 1 (Wed)
		Item 3 (Wed)	Item 2 (Wed)
			Item 3 (Wed)
Thu	Item 4 (Thu)	Item 3 (Thu)	Item 2 (Thu)
		Item 4 (Thu)	Item 3 (Thu)
			Item 4 (Thu)

Table 7.12 Illustration of duplication due to overlapping

Each component in a chain similarly contributes to the overlap in the timestamp range during harvesting. With frequent harvesting and/or frequent updates at the source, this will result in greater duplication of data. Selecting a harvesting granularity that is as small as possible (minimizing g) will minimize the duplication due to chaining of components. Fortunately, in going from version 1.1 to 2.0 of OAI-PMH, granularity can be as little as a second, instead of a day.

In all experimental systems, a granularity of seconds was used wherever possible.

7.5.8 Consistency

Suppose that $t(A)$ is the time, in seconds, between harvesting operations performed by component A. This implies that component A is at most $t(A)$ seconds out of synchronization with the source from which it is harvesting data. Now, suppose that component B harvests the same data from component A, at an interval of $t(B)$ seconds. Then component B is at most $t(B)$ seconds out of synchronization with component A, and transitively, $t(A)+t(B)$ seconds out of synchronization with the original source archive.

Similarly, each archive in a chain contributes its harvesting interval to the delay in propagating changes. Where changes have to propagate more quickly within a DL, it is desirable to use smaller harvesting intervals. However, since smaller harvesting intervals cause more network requests, when network bandwidth is a limiting factor, delays in propagation are a necessary compromise.

In the new CSTC system, consistency is maintained by explicit and controlled harvesting when changes need to be propagated. Thus, if a new item is submitted, the DBUnion is asked to harvest from its source. When the operation completes, both IRDB and DBBrowse are triggered to harvest from DBUnion. This ensures immediate updating and prevents consistency problems.

7.5.9 Network Bandwidth

Network bandwidth is not a serious problem if components are collocated on a machine or are in near proximity to one another. However, if some components are used remotely (this was tested with CSTC initially using a remote DBBrowse component), this can easily become a bottleneck.

In general, since component interactions need to be minimized and all component interactions happen over high-level network protocols, minimizing network bandwidth is already an issue that needs to be addressed. The contents of packets as well as the structure of communication can be optimized.

Thus far, no work has been done on compressing individual packets of data. However, the new OAI-PMH v2.0 supports HTTP-level compression and this can be exploited in future versions of ODL protocols.

In terms of minimizing communications, caching has the most benefit by eliminating duplication of the same requests in a given time period. Other factors considered include the algorithms for harvesting. By harvesting n records with m metadata formats as separate operations, the harvester can use m **ListRecords** requests; on the other hand, harvesting n records with m metadata formats using **ListIdentifiers** and **GetRecords** will require $(1 + n m)$ requests. These are representative of the two harvesting algorithms used by ODL components, as discussed below.

7.5.9.1 Complexity of Harvesting

The first algorithm uses **ListRecords** to maximize speed, and the second uses a combination of **ListIdentifiers** and multiple **GetRecords** requests to maximize consistency for archives with multiple metadata formats per identifier. The following analysis calculates the difference in numbers of network requests and amount of data transferred for each algorithm.

Let the time (in days) between harvesting operations = t

Let the average number of records harvested in one operation = n

Define a batch as the group of records or identifiers sent in a response before a *resumptionToken* is issued. Let the maximum size of a batch = k

Then, the average number of batches in a harvest operation, $b = \lceil n / k \rceil$

Let the average size (in bytes) of an OAI response header (XML namespace information, *responseDate*, *requestURL*, and containers) = h

Let the average size (in bytes) of a record = R

Let the average size (in bytes) of an identifier = I

Harvesting Algorithm A

If we choose a harvesting algorithm that uses a single **ListRecords** request to transfer records, then

the number of network requests

= number of batches

= b

= $\lceil n / k \rceil$

and the quantity of data transferred

= size of header + (size of record * number of records)

= $h + nR$

Harvesting Algorithm B

If we choose a harvesting algorithm that first issues **ListIdentifiers** and then issues **GetRecord** for each record,

the number of network requests

= number of batches for **ListIdentifiers** + number of **GetRecord** requests

= $\lceil n / k \rceil + n$

and the quantity of data transferred

= size of **ListIdentifiers** header + (size of identifier * number of records) + number of records * (size of **GetRecords** header + size of record)

= $h + nI + n(h + R)$

= $(h + nR) + n(I + h)$

Thus, the number of network requests as well as the quantity of data transferred will be higher for the second algorithm. The simpler first algorithm is selected for most harvesting operations to avoid this performance penalty.

7.6 SUMMARY

It was shown in this chapter that the ODL component composition approach to building DLs is understandable and simple from the perspective of most novice users with only a basic understanding of OAI and ODL.

Then it was demonstrated that the use of ODL components does not lead to intrinsic execution speed penalties – any penalties observed are due to specific tools used in the experimental systems. Some optimizations for the underlying technology and the ODL protocol design were tested and discussed.

Various projects have successfully adopted and used the ODL approach and, in particular, early ODL components. In some cases these were extended or layered for more functionality.

Lastly, some issues in storage and network bandwidth conservation were presented along with the effect these have had on the construction of recent ODL-based systems.

Chapter 8

FUTURE WORK

8.1 INTRODUCTION

This chapter discusses a number of outstanding issues that were raised during the design, testing, and implementation phases. In many instances where temporary solutions were adopted, there are more effective and efficient alternatives. Following is a list of the issues, why they were considered problematic, how they were worked around, and what improvements can be made in the future.

8.2 HARVESTING

8.2.1 Issues with Multiple Sources

Harvesting of metadata depends on the constant network availability of the data provider, or at least availability each time harvesting is performed. Experience with the ETD Union Catalog has shown that there are sometimes failures due to network outages and hardware and software issues with servers. At the OAI central registry there were 115 registered data providers as of 23 October 2002 – of these, 20 failed to respond to an **Identify** request that day. In many cases running an OAI data provider is an auxiliary task that is not as important as the primary mission of the organization. Thus, failures are not quickly attended to and service providers have to deal with this problem.

The harvester module currently in use in many ODL components will attempt to detect errors and will terminate harvesting if necessary. Since the harvesting schedule is only updated after the operation completes, a lock file is maintained to prevent two different processes from harvesting the same archive simultaneously. This lock file is removed when there is a critical error so that harvesting of a particular archive is retried the next time the harvester is run. The problem with this approach is that server failures that are not attended to for a long period of time will result in repeated harvesting failures. As a possible solution, the harvesting algorithm can keep track of the number of recent failures and employ an exponential back-off algorithm to gradually increase the time between harvesting attempts so as not to flood a server that is not responding correctly. Another useful feature that can be added is automatic email to the administrator of the server if the harvesting operation has failed for a specified amount of time.

Currently, a log of all harvesting operations is either stored locally or sent to the administrator of the component by email. This includes reporting of error conditions. For the ETD Union Catalog, this has resulted in a significant quantity of email everyday – on the order of 1 message each time a component harvested from another component. For the CSTC system(s), email notification is not used but the data is logged to local files

instead. To streamline this process, a summary email can be sent to the administrator, listing all recent activity. As a further step, this information can be made available on a website which both keeps track of the history of harvesting by one or more components and allows manual interaction with components to trigger immediate harvesting, temporarily turn off harvesting for specific archives, etc.

8.2.2 Issues with Single Sources

ODL components sometimes use harvesting to transfer a stream of data from one component to another. For some projects, a high degree of data consistency is required. In the production CSTC system, this was one of the first user complaints – that items that were accepted in the reviewing system were not immediately visible in the “search and browse” user interface. This happened because the data was not being harvested immediately by the DBBrowse component. As a temporary solution, harvesting is set to occur every 15 minutes. This does, however, result in much unnecessary harvesting of the source archive. As an alternative, the new CSTC system uses explicit harvesting, where the user interface “glue” initiates harvesting whenever an update is made. First, the DBUnion component harvests from the source; then the DBBrowse and IRDB components are made to harvest from the DBUnion. This is still far from ideal. A machine interface can be used to control the harvesting operation. A simple component can have two external interfaces: one to access the data through XOAI and a second to control the harvesting operation. Then a component will define not just which nodes it gets data from but also the nodes it sends data to. These two-way links will allow a component to trigger harvesting from its dependent nodes without any necessary interaction at the user interface layer.

An alternative approach is to factor out harvesting from all ODL components. Instead of bundling a component with a harvester, the harvester can be installed externally. Then, the harvester can obtain a stream of records from the source component and submit them to the destination component using **PutRecord** requests. This will result in a cleaner separation of functionality so that harvesting can be performed between any two arbitrary components. Also, it will enable easy replacement of the harvester.

8.3 USER INTERFACES

8.3.1 User Interaction API

Rather than hand-coding the different sets of user interface interactions, as explained previously, it may be possible to create general-purpose user interaction procedures. For example, portions of an HTML interface can be generated by submitting GetRecord requests to components and then transforming their responses using XSL stylesheets. These operations can be simplified to procedures such as:

```
$ODL_Rating->Display_GetRecord ($identifier,  
$metadataPrefix, $xsl)
```

In this command, *\$ODL_Rating* refers to the object corresponding to an instance of the DBRate component, *\$identifier* and *\$metadataPrefix* uniquely identify the record requested, and *\$xsl* is the stylesheet used to transform the response.

Similar procedures can be defined for **ListRecords**, **ListSets**, and **ListIdentifiers**.

However, for submission of records to a component (e.g., submission of a rating) via **PutRecord**, an XML fragment must be created. This can be handled by a specific client-side API to interface with an ODL component. The libraries that implement these APIs can then handle encoding and XML namespace/schema issues. For example, a rating might be submitted from the client programming language using a command of the form:

```
$ODL_Rating->Submit ($subject, $object, $rating)
```

In the above example, *\$ODL_Rating* refers to the object corresponding to an instance of the DBRate component, *\$subject* is the identifier of the user submitting the rating, *\$object* is the identifier of the resource being rated, and *\$rating* is the numerical value of the rating.

If the parameters are themselves fragments of XML or XPath specifications, it may be possible to eliminate the procedural portions of the client program. Then, user interface interaction can be controlled by a declarative specification, driven by a user interface engine.

8.3.2 MEdit Generalization

The MEdit module discussed previously is used to dynamically generate HTML forms for metadata entry and editing. The appearance of the HTML forms is fixed in the MEdit module but this can be made customizable through parameters to the module. Alternatively, the form display can be generated in UIML – User Interface Markup Language (Phanouriou, 2000) – and then transformed into appropriate displays for different projects and even different devices.

While extensions have been devised to support some of the common input types used in forms, this can be further generalized to encompass all options available within HTML. In addition, optional HTML attributes, such as CSS styles, can be specified in the schema.

Currently there is only support for a subset of the XML Schema language, but additional types and structures can be added to allow for more general XML structures with stricter type validation. Examples of types which can be added and that are popular in schema files are “anyURI” and “dateTime” – these can be used for type-validation, with additional restrictions placed through extensions if necessary. An example of such a restriction is that only dateTime values expressed in GMT are acceptable in the OAI-PMH.

While the prototype module was created in Perl, the XML Schema extensions are sufficiently general to support the creation of engines in other languages. Thus, a given schema file can produce the same interface on the same device irrespective of the engine and programming language used to create the interface. In addition, an alternative engine can be created to enter metadata one field at a time through a command-line interface. This will be useful to support configuration of the components based on an XML schema.

8.3.3 Component Composition GUI

As discussed previously, an observed trend during the component composition user study was that many participants were confused by the myriad of baseURLs used to connect components together. It may be possible to eliminate having to type those in by using a visual component composition interface similar to that used in the Microsoft Visual and Borland Builder product suites or the experimental D2K component composition tool developed at NCSA (Auvil and Clutter, 2002).

To exploit the familiarity users have with popular visual programming interfaces, it may be possible to use the existing environments to compose ODL components. Components can be mapped to applications such that either the application configures the component when run or the act of manipulating the component in the visual environment triggers a side-effect of configuring the component. Parameters can be set as property lists and sanity checks can be mapped to procedures associated with individual properties. To link the GUI and the Web server, the developer can work on the server by remotely mounting its filesystem or a client-server protocol can be developed to support remote component instantiation and configuration.

Alternative interfaces may be built from scratch, using GUI toolkits, JavaBeans container technology, or even HTML websites. The last option will allow configuration of a digital library through a Web interface, thus supporting online component configuration and composition and eliminating the need to use shell-level manipulation. Such an approach will still require installation of the component, but this can be automated if the component configuration interface also includes the ability to download components from remote sites and install them in the background.

8.3.4 Portals

Projects such as uPortal (JASIG, 2002) are developing portal technology to integrate a suite of services into a single Web user interface in a componentized fashion. ODL components are a prime example of services that can be integrated with uPortal, where each ODL service can be constructed as a uPortal channel. Thus, the portal channel will form the user interface analogue to the service-level ODL component.

There are many other portal projects – integration should be possible with any that support the construction and submission of HTTP requests and the translation of XML responses into sections of the user interface.

8.4 LOGGING

Logging of transactions occurs at multiple levels with the ODL components. Firstly, since all requests are submitted through the Web server, the server log files contain information about the components being accessed, the time of access, the source of the access, and the parameters if the request is sent using HTTP GET. If the request is sent using HTTP POST, the parameters may be logged separately if the Web server is configured to do this.

Component operations that do not involve the ODL interface will not be logged through this mechanism. One typical such operation is harvesting of metadata from source archives, performed by components such as IRDB and DBBrowse. These operations are logged independently with general information about the harvesting operation and a list of the records processed by the component. A fragment of the DBBrowse log file is displayed in Figure 8.1.

```
Tue Oct 1 11:07 : Harvesting:
Tue Oct 1 11:07 :   archive      = cstcddbrowse
Tue Oct 1 11:07 :   url          =
http://oai.dlib.vt.edu/~hussein/cgi-
bin/cstc/DBUnion/cstcdbunion/union.pl
Tue Oct 1 11:07 :   interval     = 86400 days
Tue Oct 1 11:07 :   metadataPrefix = resource
Tue Oct 1 11:07 :   set           =
Tue Oct 1 11:07 :   daysoverlap   = 1
Tue Oct 1 11:07 :   interrequestgap = 10
Tue Oct 1 11:07 :
Tue Oct 1 11:07 : [1] Processing : oai:rmetadata:2
Tue Oct 1 11:07 : [1] Indexing: oai:rmetadata:2, resource
Tue Oct 1 11:07 : [1] Done with : oai:rmetadata:2
Tue Oct 1 11:07 : [2] Processing : oai:rmetadata:3
Tue Oct 1 11:07 : [2] Indexing: oai:rmetadata:3, resource
Tue Oct 1 11:07 : [2] Done with : oai:rmetadata:3
Tue Oct 1 11:07 : Done harvesting.
```

Figure 8.1 Fragment of DBBrowse log file

First, the parameters for the harvesting operation are listed. Then, for each record encountered, entries are created when the record is obtained, when it is submitted for processing by the component (in the case of DBBrowse this involves indexing by specific fields), and when processing is completed. Finally, an entry is created to indicate that harvesting has completed.

While the non-standard log file format assists in detecting errors and confirming correct operation of the component, log file analysis tools will not operate on the data without pre-processing to transform it into a standard format, such as Extended Common Log Format (ECLF). Future component implementations will benefit from using a standardized log format, as well as logging all activity independently of the Web server.

ECLF logs basic information about each transaction since it is aimed towards Web access. A more general log format will allow the logging of client sessions and operations specific to individual components. The recently proposed XML Log File

Standard (Gonçalves, et al., 2002) may be a framework that can be applied to ODL components to capture this vital information. While the new standard is geared towards discovery services such as searching and browsing, it can be generalized to include support for inter-component interaction.

8.5 SECURITY

Network security concerns are compounded when a Web-based system is built in a componentized fashion since each component is a potential point of attack. A component may be compromised in at least two ways: by sending an abnormal request to the component, e.g., causing remote buffer overflow, or by sending a legally formatted request to a remote component that should not be accessible.

The former case is best avoided by vigilance. Remote buffer overflows and similar hacks constantly plague server administrators and require regular patching of server software and reconfiguration of servers to exclude unwanted access (Hsiao, 2001). Individual languages also have mechanisms in place to help thwart hacking attempts. Perl supports optional taint-checking (Siever, et al., 1999), where parameter values passed to any program may not be used in risky operations, like process spawning, without first being checked and transformed. This prevents the abuse of HTML forms to execute arbitrary code on Web servers.

Legally formatted requests pose a different form of threat altogether. It is conceivable that, with open standards and protocols, a malicious user might send a request to an unprotected component to delete or modify records. Using access control lists can prevent this, as was done with the Box and DBReview components, while even stronger measures like passwords may be needed to protect against breakins to trusted systems. Other components need similar mechanisms to screen out unauthorized transactions. More sophisticated rights management can be employed to finely control the interactions among components. The XrML language (ContentGuard, 2001) and the emerging Open Digital Rights Language (Ianella, 2002) can be used with Web services and data transferred over Web services and thus may be good candidates for controlling access to ODL components.

A combination of both of these approaches is necessary to minimize security risks. For the existing ODL components, some thought has been given to security but a thorough audit and integration of preventative security features is desirable before widespread production use of the components occurs.

8.6 NEW PROTOCOLS AND COMPONENTS

The protocols that were designed and the corresponding reference implementations that were built during this study are representative of the kinds of digital library services that are needed in production systems. However other services may be needed.

New protocols can be designed for the following services:

- Summarization of search results – where a categorical summary of the search results is presented alongside the search results, as is done in the ARC system (Liu, et al., 2001), or in sequence with the search results, as is done in the Amazon.com system.
- Storage and retrieval of activity history.
- Further generalization of the ODL-Rate and ODL-Review protocols to support different workflow models, e.g., an open review system where anyone may submit reviews.

Further components can be designed and built to perform the following services using existing protocols (the protocol/s that the component will support is/are indicated in parentheses):

- Automatically categorize records and insert the categories into specific fields – a metadata record mutator (OAI-PMH).
- Dynamically or statically convert metadata from one format to another (OAI-PMH).
- Support different search and browse query languages (ODL-Search, ODL-Browse).
- Implement a high performance search engine using commodity open source or commercial tools (ODL-Search). Student groups working on class projects have already done some work to illustrate the feasibility of this with the Lucene (Goetz, 2000) and Swish-E (Tennant, 2002) search engines.

Components also may be built to support multiple ODL protocols. For example, a single component can harvest and index records for the purposes of searching and browsing. This will lead to improvements in network bandwidth usage and can result in storage savings from internal sharing of data structures.

8.7 TESTING

Testing of components is done at various different layers within the internal component architecture, but always using hand-coded tests – this was discussed in the prior chapter on testing.

Further standardization of the software libraries used in development can lead to a tool suite that will not only be adaptable to future versions of the OAI and ODL protocols, but also possibly to other client/server protocols. While XML tools are still not widely deployed and are notorious for high degrees of complexity, future combinations of XSD (Schema), XSLT (Transformation), and XPath (Path Specifications) can make protocol testing a fully automated process for emerging client/server protocols using XML as the underlying technology. The Repository Explorer and Component Explorer both already use XML Schema for validation of response formats. Further checking can be done using other tools such as Schematron (Dodds, 2001), which can test for internal relationships among tags in the XML document. Finally, instead of manually parsing the responses, the user interface can be constructed by transforming the response into HTML using an XSL stylesheet. Responses to single service requests thus can be processed completely using XML tools driven by declarative specifications, independently of the protocol being tested.

For automatic testing, it is necessary to specify a sequence of service requests, where some are parameterized by data from earlier responses. As an example of this, the Repository Explorer tests for a correct response to **ListMetadataFormats** using an *identifier* obtained from an earlier response to **ListIdentifiers**. This can be generalized using a declarative specification language to indicate the sequence of service requests and their parameters. The following is a simple example of such a specification:

```
<request label="1" xsd="OAI-PMH.xsd">
  <parameter name="verb">ListIdentifiers</parameter>
</request>
<request label="2" xsd="OAI-PMH.xsd">
  <parameter name="verb">ListMetadataFormats</parameter>
  <parameter name="identifier" type="xpath" source="1">
    OAI-PMH/ListIdentifiers/header/identifier
  </parameter>
</request>
```

In this example, the first request submitted is **ListIdentifiers**. The second request is **ListMetadataFormats**, which uses an XPath expression to extract a parameter from the first response. Both responses will be validated against the “OAI-PMH.xsd” schema.

Using such a declarative specification as input to a “protocol-testing engine”, it will be possible to perform tests analogous to the Repository Explorer’s automatic tests. The advantages of this approach are that different OAI and ODL protocols can be tested using a single program driven by specialized declarations for each protocol. Such scripting will facilitate performance testing as well.

These techniques also can be generalized to support other XML-based protocols, including those related to Web Services.

8.8 INSTALLATION AND REGISTRATION

As discussed previously, system designers do not want to remember a multitude of baseURLs corresponding to each service component within a system. To make the task of component composition simpler, these baseURLs can be hidden behind a graphical user interface, which tracks the names of components and associates baseURLs with them automatically when needed.

As a step further, any environment for composing components can keep a central – at least for a project or server – registry of all installed components. When each new component is installed and configured, it is registered and whenever a component is being connected to another component, the registry can be queried for a list of components to connect to.

Registries can be built at both the template and instance level. With support for both, each template can be registered at the point of installation. Then, whenever an instance is

configured, that can be registered as well. Composition of components then will have access to the list of configured instances, with the option to configure a new instance if necessary.

A central registry of templates can be maintained so that each remote registry updates its list of available templates at regular intervals. It may be possible to download and install component templates automatically if platform-independent bytecode is used such as in the Java and Jython languages or if the language is purely scripted such as in the case of Perl. Such a delivery mechanism also can be used to distribute module updates.

Further thought needs to go into how best to design such a system to be as independent of platform and language as possible. Also, additional security and accounting concerns will need to be addressed as the registration system becomes more complex and interactive. The UDDI project (Ariba, Inc., et al., 2000) is attempting to build registries of Web Services for cross-organization collaboration. Some of their formal definitions of services also may be applicable to the fine-grained ODL inter-component interaction.

8.9 PERFORMANCE

Any interaction with a Web server will have upper bounds on performance – at some point the load can only be lessened by using more server resources. This includes the possibilities of faster servers, servers with more memory, and faster internal bus speeds.

Another option is to distribute the load across multiple servers. Since a typical interaction with an ODL-based system involves internal communication among components, these components can be distributed among multiple machines, thereby decreasing the load and specializing the function of each machine. When using a site-wide registry, as discussed above, there will be no difference in operation except the possible penalty incurred from network communication. High-speed special-purpose networks can reduce these delays. Partitioning of the components into sets using a minimum-flow algorithm also will lessen the effect.

As digital library systems become even more popular, components can be duplicated on multiple systems with transparent round-robin DNS resolution to distribute the load among the different instances – forming a “component farm” analogue to the “server farm” used for many large websites. For more sophisticated solutions, a load balancing algorithm can be deployed to distribute components across institutions or sites using the framework of the Internet2 Distributed Storage Initiative (Beck and Moore, 1998).

Thoroughly analyzing the protocol specifications and removing redundancy and irrelevant information can result in additional low-level performance improvements. For example, while the *timestamp* within each OAI record is indispensable for harvesting, it is not used by most components when creating user-directed displays. In addition, every response has the *requestURL* embedded and it is never used except for debugging purposes. A complete profiling of the use of other features can expose which features are really useful for the base XOAI protocol, which can be relegated to higher levels, and which can be eliminated altogether from the ODL protocols.

8.10 NEW STANDARDS

8.10.1 OAI-PMH v2.0

Since work began on the ODL component framework, the OAI has released version 2.0 of the OAI-PMH. Some of the new features in the protocol influenced by early ODL work and other projects are:

- Finer datestamp granularities to support times in addition to dates.
- The addition of optional attributes to indicate the *cursor* and *completeListSize* for a response.
- Association of *datestamp* with individual metadata records and not with items, thus giving more independence to metadata records.
- A well-defined mechanism to return errors and exceptions in XML format, thus separating the OAI protocol from HTTP.
- The elimination of timezones altogether – all datestamps now have to be in Greenwich Mean Time. This removes the need to handle both differences in timezones and differences in daylight savings policies.

Some XOAI-PMH features were proposed to the OAI Technical Committee but they were not pursued because they did not have general appeal. Specifically, the **PutRecord** service request did not appeal to members who wanted to avoid the security concerns inherent with 2-way data transfer; and the idea of a response-level container did not get much support because the protocol was not being designed for extensibility.

A few of the new features of OAI-PMH v2.0 are problematic for ODL. The requirement that each record lists the sets it is a member of in its header will not work for ODL protocols where the *set* parameter is used to encode complex expressions. For example, ODL-Search cannot list every search query that a record will satisfy. Also, the idempotence of requests, which is now explicitly required, cannot be satisfied if the protocol is not read-only.

Ultimately, if a new version of the XOAI protocol is derived from OAI-PMH v2.0, there will be a different set of changes necessary from those that were applied to OAI-PMH v1.1.

8.10.2 SOAP-OAI and SOAP-ODL

During the process of designing and experimenting with OAI-PMH v2.0, there emerged the possibility of using SOAP (Box, et al., 2000) instead of HTTP as the underlying layer. The primary advantage of SOAP to OAI is that it provides an XML-based parameter passing convention that is now widely supported. This will eliminate the need for CGI and the different encodings of CGI and XML data, as well as provide the facility for Schema-based validation of service request parameters.

While SOAP was not adopted by the OAI because of time constraints and because the standard is not yet stable, some changes have been made to eventually support a move

towards SOAP. The schemata for OAI-PMH responses were unified and errors are now handled through XML responses. The former was suggested by experiences from preliminary work in porting OAI-PMH to SOAP. From this preliminary work, an example SOAP service request body that can correspond to **GetRecord** is shown in Figure 8.2.

```
<?xml version="1.0"?>

<GetRecordRequest
  xmlns="http://oai.dlib.vt.edu/OAI/1.1/SOAP/1.1/OAI_GetRecord_Request"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://oai.dlib.vt.edu/OAI/1.1/SOAP/1.1/OAI_GetRecord_Request
    http://oai.dlib.vt.edu/OAI/1.1/SOAP/1.1/OAI_GetRecord_Request.xsd"
>

  <identifier>test</identifier>
  <metadataPrefix>oai_dc</metadataPrefix>

</GetRecordRequest>
```

Figure 8.2 Example of SOAP GetRecord service request body

If OAI opts to go the SOAP route, newer versions of ODL can build on it and inherit the advantages of simpler parameter passing. In addition, each ODL protocol can have its own schema for parameter passing instead of overloading the standard OAI-PMH parameters.

8.10.3 ODL v2.0

In previous chapters it has been demonstrated that systems can be built using ODL components and that these systems can be both effective and efficient. The premise of ODL has always been simplicity, and this drove the approach of building on the OAI protocol rather than designing a new protocol from scratch.

In many cases this has meant overloading the meaning of OAI-PMH service requests and ignoring service requests which made no sense, for example, **ListSets** for ODL-Search. The base XOAI protocol thus contains some features which are not necessary in some of the higher-level ODL protocols.

Another experience from designing and implementing the ODL protocols has been that many of the protocols share common requirements, like the number of *hits* encoded in the *responseContainer* and the *start* and *stop* parameters encoded in *set* when requesting a list of records.

These experiences suggest that a newer protocol can be designed to revert to the Kahn-Wilensky model of data access as an operation distinct from metadata harvesting. Instead of building a data access model over the OAI protocol, it may be possible to create an abstract data access protocol as a lower layer beneath the OAI protocol. Metadata harvesting will then be a higher-level operation on the same level as the rest of the ODL protocols. This will have the net effect of:

- Removing the unnecessary harvesting-specific information from ODL protocols.
- Eliminating the need to define service requests where they are not applicable.
- Creating a more general model to support future digital library services, independent of changes to the OAI-PMH.
- Separating ODL from OAI to decrease confusion about their differences, while still exploiting their similarities.

Figure 8.3 illustrates the differences between the current ODL approach and the one being proposed.

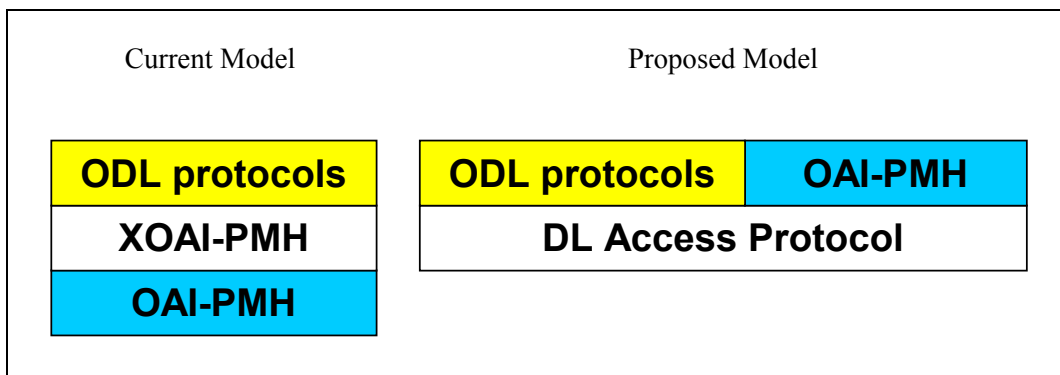


Figure 8.3 Current and proposed models for OAI/ODL relationship

The proposed DL access protocol can have optional primitive operations to

- Identify the component.
- Retrieve a single item.
- Retrieve multiple items based on starting and stopping positions.
- Submit a single item.

Extensibility can be built into the protocol to support the addition of service requests and parameters for higher-level ODL and OAI protocols.

Chapter 9

CONCLUSIONS

Building digital libraries is not a simple process – or so say those who build them.

This work has attempted to dispel that myth by proposing and testing a component framework to support the construction of digital libraries in a simple and repeatable fashion.

The Open Archives Initiative set the stage for large-scale interoperability of DL systems by specifying and supporting the Protocol for Metadata Harvesting. Open Digital Libraries generalize the notions of the PMH to support fine-grained access to components, using an extension of the protocol (XOAI-PMH) from which specific service-directed ODL protocols are derived.

Protocols have been specified for many popular DL services. Reference implementations of each were created. These were then integrated into multiple existing and built-from-scratch systems to demonstrate their ability to integrate with other components and legacy systems with differing requirements and architectures.

The experimental systems have shown that:

1. ODL components can be created as self-contained configurable entities.
2. It is possible to construct DL interfaces using a network of ODL components to provide the back-end processing.
3. The DL interfaces created are not distinguishable from monolithic systems.
4. ODL components are extensible and reusable. Some portability and independence of components can be achieved.
5. Performance is very important but there are workable solutions to address these issues without sacrificing generality.

A user study was conducted and performance measurements were taken. These supported the assertions that:

1. ODL is relatively simple and understandable.
2. ODL layering does not negatively impact the performance of systems.
3. Component composition does not negatively impact the performance of user interfaces.

These experiments have shown that there is promise for ODL and similar approaches to replace the traditional monolithic digital library systems. The results from this work

vindicate the Web Services approach to building systems by confirming the efficacy and efficiency of such systems. Also, by investigating particular services and the semantics of service-level protocols, this work has demonstrated the applicability of Web Services to digital libraries. As the Web Services technology stabilizes, it will play a vital role in the future of distributed digital libraries.

It is hoped that the results of this work will change the way people build Digital Libraries. The evaluations and feedback received from users and colleagues has strengthened the case for building DL access protocols to support high-level services, and composing of those services into complete Digital Library systems.

Building upon a foundation of extensibility, it then will be possible for DL researchers and practitioners to work on providing more interesting services to users, thus bridging the wide gap between current research and production systems, and ultimately making information more accessible to people.

REFERENCES

- ACME Labs (2002), thttpd - tiny/turbo/throttling HTTP server. Website <http://www.acme.com/software/thttpd/>
- Amazon.com (2002), *Internet Movie Database*. Website <http://us.imdb.com>
- ANSI/NISO (1995), *Information Retrieval (Z39.50): Application Service Definition and Protocol Specification (ANSI/NISO Z39.50-1995)*, Bethesda, MD: NISO Press.
- Apache Software Foundation, The (2002), *Xerces Java Parser*. Website <http://xml.apache.org/xerces-j/index.html>
- Apparao, Vidur, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacobs, Arnaud Le Hors, Gavin Nicol, Jonathan Robie, Robert Sutor, Chris Wilson, and Lauren Wood (editors) (1998), *Document Object Model (DOM) Level 1 Specification*, W3C, 1 October 1998. Available <http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/>
- Ariba, Inc., IBM and Microsoft (2000), *UDDI Technical White Paper*, 6 September 2000. Available http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf
- Atkins, Anthony (2002), *Resources for Developers of ETD databases*. Website <http://scholar.lib.vt.edu/ETD-db/developer/>
- Atkins, Anthony, Edward A. Fox, Robert K. France, and Hussein Suleman (2001), *ETD-ms: an Interoperability Metadata Standard for Electronic Theses and Dissertations version 1.00*, NDLTD. Available <http://www.ndltd.org/standards/metadata/>

- Auvil, Loretta, and David Clutter (2002), *Data to Knowledge (D2K) Tutorial*, UIUC, 18 March 2002. Available <http://www.ncsa.uiuc.edu/Divisions/DMV/ALG/D2K-tutorial.htm>
- Baldonado, M., C. K. Chang, L. Gravano, and A. Paepcke (1997), "The Stanford Digital Library Metadata Architecture", in *International Journal on Digital Libraries*, Vol. 1, No. 2, pp. 108-121. Available <http://www-diglib.stanford.edu/cgi-bin/get/SIDL-WP-1996-0051>
- Beck, M., and T. Moore (1998), "The I-2 DSI Project: An Architecture for Internet Content Channels", in *Computer Networking and ISDN Systems*, Vol. 30, No. 22-23, pp. 2141-2148.
- Berners-Lee, Tim (1996), *The World Wide Web: Past, Present and Future*, W3C. August 1996. Available <http://www.w3.org/People/Berners-Lee/1996/ppf.html>
- Berners-Lee, Tim and Mark Fischetti (1999), *Weaving the Web*, Harper, San Francisco.
- Berners-Lee, Tim, James Hendler, and Ora Lassila (2001), "The Semantic Web", in *Scientific American*, May 2001. Available <http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html>
- Berry, Michael W., and Murray Browne (1999), *Understanding Search Engines: Mathematical Modeling and Text Retrieval*, SIAM, Philadelphia.
- Bertocco, Sara (2001), "Torii, an Open Portal over Open Archives", in *High Energy Physics Libraries Webzine*, Issue 4, 25 June 2001. Available <http://library.cern.ch/HEPLW/4/papers/4/>

- Bird, Steven, and Gary Simons (2001), “The OLAC Metadata Set and Controlled Vocabularies”, in *Proceedings of the ACL/EACL Workshop on Sharing Tools and Resources for Research and Education*, Toulouse, July 2001, Association for Computational Linguistics. Preprint <http://arXiv.org/abs/cs/0105030>
- Birmingham, W. P. (1995), “An Agent-Based Architecture for Digital Libraries”, in *D-Lib Magazine*, Vol. 1, No. 1, July 1995. Available <http://www.dlib.org/dlib/July95/07birmingham.html>
- Bowman, C. M., P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz (1995), “The Harvest Information Discovery and Access System”, in *Computer Networks and ISDN Systems*, Vol. 28, pp. 119-125.
- Box, Don, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer (2000), *Simple Object Access Protocol (SOAP) v1.1*, W3C, 8 May 2000. Available <http://www.w3.org/TR/SOAP/>
- Brain, Marshall (2002), *How Stuff Works*. Website <http://www.howstuffworks.com>
- Bray, T., J. Paoli, C. M. Sperberg-McQueen, and Eve Maler (editors) (2000), *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C. Available <http://www.w3.org/TR/2000/REC-xml-20001006>.
- Brewer, Eric A. (2001), “When Everything is Searchable”, in *Communications of the ACM*, Vol. 44, No. 3, March 2001, pp. 53-54.
- Brown, M. R. (1996), “FastCGI – A High-Performance Gateway Interface”, position paper at *Programming the Web - a search for APIs* workshop, Fifth International

- World Wide Web Conference, Paris, France, 6 May 1996. Available <http://www.fastcgi.com/devkit/doc/www5-api-workshop.html>
- Castelli, Donatella, and Pasquale Pagano (2002), “OpenDLib: A Digital Library Service System”, in *Research and Advanced Technology for Digital Libraries*, Proceedings of the 6th European Conference, ECDL 2002, Rome, Italy, September 2002, pp. 292-308.
- Clark, James (1999), *XSL Transformations Version 1.0*, W3C, 16 November 1999. Available <http://www.w3.org/TR/xslt>
- Clark, James, and Steve DeRose (editors) (1999), *XML Path Language (XPath) Version 1.0*, W3C, 16 November 1999. Available <http://www.w3.org/TR/xpath>
- CLIR (1998), *CPA Annual Report: 1997-1998*, Council on Library and Information Resources. Available <http://www.clir.org/pubs/annual/annrpt97/libraries.html>
- ContentGuard (2001), *XrML 2.0 Technical Overview*. Available <http://www.xrml.org/reference/XrMLTechnicalOverviewV1.pdf>
- Cooper, Clark (1999), “Using Expat”, on *O’Reilly’s xml.com*, 1 September 1999. Available <http://www.xml.com/pub/a/1999/09/expat/index.html>
- Crispin, M. (1996), *RFC2060: Internet Mail Access Protocol – version 4rev1*, Network Working Group, December 1996. Available <ftp://ftp.isi.edu/in-notes/rfc2060.txt>
- Davis, James R., and Carl Lagoze (2000), “NCSTRL: Design and Deployment of a Globally Distributed Digital Library”, in *JASIS*, Vol. 51, No. 3, pp. 273-280.
- DCMI (1997), *Dublin Core Metadata Element Set Version 1.1: Reference Description*. Available <http://www.dublincore.org/documents/dces/>

DCMI (2000), *Dublin Core Qualifiers*, Dublin Core Metadata Initiative, 11 July 2000.

Available <http://www.dublincore.org/documents/dcmes-qualifiers/>

Dijkstra, Edsger (2001), “The End of Computing Science”, in *Communications of the ACM*, ACM, Vol. 44, No. 3, March 2001, p. 92.

Dodds, Leigh (2001), *Schematron: validating XML using XSLT*, XSLT-UK Conference, Oxford, England, 8 April 2001. Available

http://www.ldodds.com/papers/schematron_xsltuk.html

Fallside, David C. (editor) (2001), *XML Schema Part 1: Structures and Part 2:*

Datatypes, W3C, 2 May 2001. Available <http://www.w3.org/TR/xmlschema-1/>
and <http://www.w3.org/TR/xmlschema-2/>

Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee (1999), *RFC2616: Hypertext Transfer Protocol – HTTP 1.1*, Network Working Group, June 1999. Available <ftp://ftp.isi.edu/in-notes/rfc2616.txt>.

Finin, Tim, Yannis Labrou, and James Mayfield (1997), “KQML as an agent communication language”, in *Software Agents*, AAAI Press/The MIT Press.

Available <http://www.cs.umbc.edu/agents/introduction/kqmlacl.ps>

Fox, Edward A. (1999), “Networked Digital Library of Theses and Dissertations”, in *Proceedings of DLW15*, July 1999. Nara, Japan: ULIS. Available

<http://www.ndltd.org/pubs/dlw15.doc>.

Fox, Edward A. (2002), *Networked Digital Library of Theses and Dissertations*. Website

<http://www.ndltd.org>

Fox, Edward A., and Lillian Cassel (editors) (2002), *Journal of Educational Resources in Computing*, ACM. Available <http://www.acm.org/pubs/jeric/>

Fox, E. A., B. DeVane, J. L. Eaton, N. A. Kipp, P. Mather, T. McGonigle, G. McMillan, and W. Schweiker (1997), "Networked Digital Library of Theses and Dissertations: An International Effort Unlocking University Resources", in *D-Lib Magazine*, Vol. 3, No. 9, September 1997. Available <http://www.dlib.org/dlib/september97/theses/09fox.html>

Fox, E. A., J. L. Eaton, G. McMillan, N. A. Kipp, L. Weiss, E. Arce, and S. Guyer (1996), "National Digital Library of Theses and Dissertations: A Scalable and Sustainable Approach to Unlock University Resources", in *D-Lib Magazine*, Vol. 2, No. 8, September 1996. Available <http://www.dlib.org/dlib/september96/theses/09fox.html>

Fox, Edward A., Deborah Knox, Lillian Cassel, John A. N. Lee, Manuel Pérez-Quñones, John Impagliazzo, and C. Lee Giles (2002), *CITIDEL: Computing and Information Technology Interactive Digital Educational Library*. Website <http://www.citidel.org>

Fox, Edward A., Deborah Knox, Scott Grissom, and Rachelle Heller (2002), *Computer Science Teaching Center*. Website <http://www.cstc.org>

France, Robert K. (2001), *MARIAN Digital Library Information System*. Website <http://www.dlib.vt.edu/products/marian.html>

France, Robert K. (2001), *Effective, Efficient Retrieval in a Network of Digital Information Objects*, Ph.D. dissertation, Virginia Polytechnic Institute and State

University. Available <http://scholar.lib.vt.edu/theses/available/etd-11272001-124212/>

Freier, Alan O., Philip Karlton, and Paul C. Kocher (1996), *The SSL Protocol Version 3.0*, Transport Layer Security Working Group, 18 November 1996. Available <http://www.netscape.com/eng/ssl3/draft302.txt>

Gladney, H., Z. Ahmed, R. Ashany, N. J. Belkin, E. A. Fox, and M. Zemankova (1994), *Digital Library: Gross Structure and Requirements*, Workshop on On-line Access to Digital Libraries, June 1994.

Goetz, Brian (2000), *The Lucene search engine: Powerful, flexible and free*, JavaWorld. Available <http://www.javaworld.com/javaworld/jw-09-2000/jw-0915-lucene.html>

Goldberg, David, David Nichols, Brian M. Oki, and Douglas Terry (1992), “Using collaborative filtering to weave an information tapestry” in *Communications of the ACM – Special Issue on Information Filtering*, Vol. 35, No. 12, December 1992, pp. 61-70.

Gonçalves, Marcos André, Ming Luo, Rao Shen, Mir Farooq Ali, and Edward A. Fox (2002), “An XML Log Standard and Tool for Digital Library Logging Analysis”, in *Research and Advanced Technology for Digital Libraries*, Proceedings of the 6th European Conference, ECDL 2002, Rome, Italy, September 2002, pp. 129-143.

Grangard, Anders (2001), *ebXML Technical Architecture Specification v1.0.4*, ebXML, 16 February 2001. Available <http://www.ebxml.org/specs/ebTA.pdf>

Gravano, L., K. Chang, H. Garcia-Molina, C. Lagoze, and A. Paepcke (1997), *STARTS: Stanford Protocol Proposal for Internet Retrieval and Search*. Available

<http://www-db.stanford.edu/~gravano/starts.html>.

Gray, Terry (1995), *Comparing Two Approaches to Remote Mailbox Access: IMAP vs. POP*, The IMAP Connection, 18 September 1995. Available

<http://www.imap.org/papers/imap.vs.pop.brief.html>

Greef, Arthur (1998), *Partner Interface Process Technical Architecture*, RosettaNet.

Available <http://www.rosettanet.org>

Green, Noah, Panagiotis G. Ipeirotis, and Luis Gravano (2001),

“SDLIP+STARTS=SDARTS: A Protocol and Toolkit for Metasearching”, in

Proceedings of First ACM/IEEE-CS Joint Conference on Digital Libraries,

Roanoke, VA, USA, 24-28 June 2001, pp. 207-214.

Gundavaram, Shishir (1996), *CGI Programming on the World Wide Web*, O'Reilly and

Associates, 1 March 1996. Available <http://www.oreilly.com/openbook/cgi/>

Halbert, M. (2002), *AmericanSouth.org*. Website <http://www.americansouth.org>

Harnad, S. (1999), “Free at Last: The Future of Peer-Reviewed Journals”, in *D-Lib*

Magazine, Vol. 5, No. 12, December 1999. Available

<http://www.dlib.org/dlib/december99/12harnad.html>

Hsiao, Aron (2001), *Sams Teach Yourself Linux Security Basic in 24 Hours*, Sams Publishing, Indiana.

Ianella, Renato (2002), *Open Digital Rights Language (ODRL) Version 1.1*, W3C, 19

September 2002. Available <http://www.w3.org/TR/odrl/>

- IMS Global Learning Consortium, Inc. (1999), *IMS Learning Resource Meta-data Information Model*, IMS. Available
<http://www.imsproject.org/metadata/mdinfov1p1.html>
- InfoSpace (2002), *MetaCrawler Search Engine*. Website
<http://www.metacrawler.com/index.html>
- ISO (1994), *ISO/IEC 7498-1:1994, Open Systems Interconnection Basic Reference Model: The Basic Model*, International Organization for Standardization.
- JASIG (2002), *uPortal 2.0 Architecture Overview*. Available
http://mis105.mis.udel.edu/jasig/uportal/architecture/uPortal_architecture_overview.pdf
- Kahn, Robert, and Robert Wilensky (1995), *A Framework for Distributed Digital Object Services*. Available <http://www.cnri.reston.va.us/k-w.html>.
- Lagoze, C., and J. R. Davis (1995), “Dienst - An Architecture for Distributed Document Libraries”, in *Communications of the ACM*, Vol. 38, No. 4, ACM, p. 47.
- Lagoze, Carl, Walter Hoehn, David Millman, William Arms, Stoney Gan, Dianne Hillmann, Christopher Ingram, Dean Krafft, Richard Marisa, Jon Phipps, John Saylor, Carol Terrizzi, James Allan, Sergio Guzman-Lara, and Tom Kalt (2002), “Core Services in the Architecture of the National Science Digital Library (NSDL)”, in *Proceedings of Second ACM/IEEE-CS Joint Conference on Digital Libraries*, Portland, OR, USA, 14-18 July 2002, pp. 201-209.

Lagoze, Carl, and Herbert Van de Sompel (2001), *The Open Archives Initiative Protocol for Metadata Harvesting – Version 1.1*, Open Archives Initiative, January 2001.

Available <http://www.openarchives.org/OAI/1.1/openarchivesprotocol.htm>

Lagoze, Carl, and Herbert Van de Sompel (2001), “The Open Archives Initiative: Building a low-barrier interoperability framework”, in *Proceedings of the ACM-IEEE Joint Conference on Digital Libraries*, Roanoke, VA, USA, 24-28 June 2001, pp. 54-62.

Lagoze, Carl, Herbert Van de Sompel, Michael Nelson, and Simeon Warner (2002), *The Open Archives Initiative Protocol for Metadata Harvesting – Version 2.0*, Open Archives Initiative, June 2002. Available

<http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm>

LANL (2002), *arXiv.org*. Website <http://www.arXiv.org>

Lasher, R., and D. Cohen (1995), *A Format for Bibliographic Records (RFC1807)*.

Available <http://info.internet.isi.edu:80/in-notes/rfc/files/rfc1807.txt>.

Leiner, B. M. (1998), “The NCSTRL Approach to Open Architecture”, in *D-Lib Magazine*, Vol. 4, No. 11, December 1998. Available

<http://www.dlib.org/dlib/december98/leiner/12leiner.html>

Lesk, Michael (1997), *Practical Digital Libraries: Books, Bytes, and Bucks*, Morgan Kaufmann Publishers, San Francisco.

Leuf, Bo, and Ward Cunningham (2001), *The Wiki Way: Collaboration and Sharing on the Internet*, Addison-Wesley, Longman.

Levy, D., and C. C. Marshall (1995), “Going Digital: A Look at Assumptions Underlying Digital Libraries”, in *Communications of the ACM*, Vol. 38, No. 4, ACM, pp. 78-84.

Library of Congress (2002), *American Memory*. Website <http://memory.loc.gov/>

Library of Congress (2002), *MARC Standards*. Website <http://www.loc.gov/marc/>

Liu, Xiaoming, Kurt Maly, Mohammad Zubair, and Michael L. Nelson (2001), “Arc: an OAI service provider for cross-archive searching”, in *Proceedings of First ACM/IEEE-CS Joint Conference on Digital Libraries*, Roanoke, VA, USA, 24-28 June 2001, pp. 65-66.

Luo, Ming (2002), *Digital Libraries in a Box*. Website <http://dlbox.nudl.org>

Masinter, L. (1998), *RFC2324: Hyper Text Coffee Pot Control Protocol (HTCPCP/1.0)*, Network Working Group, 1 April 1998. Available <ftp://ftp.isi.edu/in-notes/rfc2324.txt>

Merriam-Webster (2002), *Merriam-Webster Online*. Website <http://www.m-w.com/>

Myers, J., and M. Rose (1996), *RFC1939: Post Office Protocol – Version 3*, Network Working Group, May 1996. Available <ftp://ftp.isi.edu/in-notes/rfc1939.txt>

Nava Muñoz, Sandra Edith (2002), *Federación de Bibliotecas Digitales utilizando Agentes Móviles (Digital Libraries Federation using Mobile Agents)*, Master’s thesis, Universidad de las Américas – Puebla.

NCSTRL (2002), *Networked Computer Science Technical Reference Library*. Website <http://www.ncstrl.org>

- NHSE (2002), *Repository-in-a-Box*. Website <http://www.nhse.org/RIB/>
- Nielsen, Jakob (2001), *Search: Visible and Simple*, Jakob Nielsen's AlertBox, 13 May 2001. Available <http://www.useit.com/alertbox/20010513.html>
- Nierstrasz, Oscar, and Laurent Dami (1995), "Component-Oriented Software Technology", in *Object-Oriented Software Composition*, edited by Oscar Nierstrasz and Dennis Tsichritzis, Prentice-Hall.
- Norman, Donald (1990), *The Design of Everyday Things*, Currency/Doubleday, New York.
- NSF (2002), *National Science, Mathematics, Engineering and Technology Education Digital Library (NSDL)*. Website <http://www.nsdlnet.org/>
- Nwana, Hyacinth S., and Divine T. Ndumu (1999), "A Perspective on Software Agents Research", in *The Knowledge Engineering Review*, Vol. 14, No. 2, pp. 1-18. Available <http://agents.umbc.edu/introduction/hn-dn-ker99.html>
- OAI (2002), *Open Archives Initiative*. Website <http://www.openarchives.org>
- OCLC, Inc. (2002), *Online Computer Library Center*. Website <http://www.oclc.org>
- OCLC, Inc. (2002), *OCLC WorldCat*. Website <http://www.oclc.com/oclc/menu/colpro.htm>
- Ogbuji, Uche (2000), *Using WSDL in SOAP applications*, IBM developerWorks, November 2000. Available <http://www-106.ibm.com/developerworks/webservices/library/ws-soap/index.html>
- OpCit (2002), *E-Prints*. Website <http://www.eprints.org/>

- Paepcke, A., R. Brandriff, G. Janee, R. Larson, B. Ludaescher, S. Melnik, and S. Raghavan (2000), "Search Middleware and the Simple Digital Library Interoperability Protocol", in *D-Lib Magazine*, Vol. 6, No. 3, March 2000. Available <http://www.dlib.org/dlib/march00/paepcke/03paepcke.html>
- Payette, S., and Lagoze, C. (1998), "Flexible and Extensible Digital Object and Repository Architecture", in *Proceedings of Second European Conference on Research and Advanced Technology for Digital Libraries*, Heraklion, Crete, Greece, 21-23 September 1998, Springer, (Lecture notes in computer science; Vol. 1513).
- Phanouriou, Constantinos (2000), *UIML: A Device-Independent User Interface Markup Language*, Ph.D. dissertation, Virginia Polytechnic Institute and State University.
- Phillips, Paul, Larry Doolittle, and Jon Nelson (2002), *Boa Webserver*. Website <http://www.boa.org/>
- Powell, James, and Edward A. Fox (1998), "Multilingual Federated Searching Across Heterogeneous Collections", in *D-Lib Magazine*, Vol. 4, No. 8, September 1998. Available <http://www.dlib.org/dlib/september98/powell/09powell.html>
- Raymond, Eric S., and Bob Young (2001), *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly and Associates, 15 January 2001.
- Resnick, Paul, and Hal R. Varian (1997), "Recommender Systems" in *Communications of the ACM*, Vol. 40, No. 3, March 1997, pp. 56-58.

RFC Editor, et al. (1999), *30 Years of RFCs*, Network Working Group, 7 April 1999.

Available <ftp://ftp.rfc-editor.org/in-notes/rfc2555.txt>

Roscheisen, M., M. Baldonado, C. Chang, L. Gravano, S. Ketchpel, and A. Paepcke (1998), “The Stanford InfoBus and Its Service Layers: Augmenting the Internet with Higher-Level Information Management Protocols”, in *Digital Libraries in Computer Science: The MeDoc Approach*, Lecture Notes in Computer Science, No. 1392, Springer, 8 August 1998. Available

<http://dbpubs.stanford.edu:8090/pub/1998-25>

Shafer, Keith, Stuart Weibel, Erik Jul, and Jon Fausey (1996), *Introduction to Persistent Uniform Resource Locators*, OCLC. Available <http://purl.oclc.org/docs/inet96.html>

Sieve (2002), *Sieve*. Website <http://simon.cs.vt.edu/sieve/>

Siever, Ellen, Stephen Spainhour, and Nathan Patwardhan (1999), *Perl in a Nutshell*, O'Reilly and Associates, Sebastopol, California.

Simeonov, Simeon (1998), “WDDX: Distributed Data for the Web”, in *Proceedings of XML '98*, Chicago, USA, December 1998. Available

<http://www.infoloom.com/gcaconfs/WEB/chicago98/simeonov.htm>

SpeedyCGI (2002). *SpeedyCGI*. Website <http://daemoninc.com/speedycgi/>

Suleman, Hussein (2001), “Enforcing Interoperability with the Open Archives Initiative Repository Explorer”, in *Proceedings of the ACM-IEEE Joint Conference on Digital Libraries*, Roanoke, VA, USA, 24-28 June 2001, pp. 63-64.

Suleman, Hussein (2002), *OAI Repository Explorer*. Website

http://purl.org/net/oai_explorer

- Suleman, Hussein, Anthony Atkins, Marcos A. Gonçalves, Robert K. France, Edward A. Fox, Vinod Chachra, Murray Crowder, and Jeff Young (2001), “Networked Digital Library of Theses and Dissertations: Bridging the Gaps for Global Access – Part 1 and 2”, in *D-Lib Magazine*, Vol. 7, No. 9, September 2001. Available <http://www.dlib.org/dlib/september01/suleman/09suleman-pt1.html> and <http://www.dlib.org/dlib/september01/suleman/09suleman-pt2.html>
- Suleman, Hussein, and Edward A. Fox (2001), “A Framework for Building Open Digital Libraries”, in *D-Lib Magazine*, Vol. 7, No. 12, December 2001. Available <http://www.dlib.org/dlib/december01/suleman/12suleman.html>
- Sun, Sam X., and Larry Lannum (2002), *Handle System Overview*, Internet Engineering Task Force, September 2002. Available <http://www.ietf.org/internet-drafts/draft-sun-handle-system-10.txt>
- Szyperski, Clemens (2000), “Component Software and the Way Ahead”, in *Foundations of Component-based Systems*, edited by Gary T. Leavens and Murali Sitaraman, Cambridge University Press, Cambridge.
- Tennant, Roy (2002), *Swish-E*. Website <http://swish-e.org/>
- Umar, Amjad (1997), *Object-Oriented Client/Server Internet Environments*, Prentice Hall, New Jersey.
- Van de Sompel, Herbert (2000), *Schema for MARC metadata format*, Open Archives Initiative. Available http://www.openarchives.org/OAI/oai_marc.xsd
- Van de Sompel, Herbert, and Patrick Hochstenbach (1999), “Reference Linking in a Hybrid Library Environment”, in *D-Lib Magazine*, Vol. 5, No. 4, April 1999.

Available http://www.dlib.org/dlib/april99/van_de_sompel/04van_de_sompel-pt1.html

Van de Sompel, Herbert, Thomas Krichel, Michael L. Nelson, Patrick Hochstenbach, Victor M. Lyapunov, Kurt Maly, Mohammad Zubair, Mohamed Kholief, Xiaoming Liu, and Heath O'Connell (2000), "The UPS Prototype: An Experimental End-User Service across E-Print Archives", in *D-Lib Magazine*, Vol. 6, No. 2, February 2000. Available <http://www.dlib.org/dlib/february00/vandesompel-ups/02vandesompel-ups.html>

Van de Sompel, Herbert, and Carl Lagoze (2000), "The Santa Fe Convention of the Open Archives Initiative", in *D-Lib Magazine*, Vol. 6, No. 2, February 2000. Available <http://www.dlib.org/dlib/february00/vandesompel-oai/02vandesompel-oai.html>

VTLS (2002), *VTLS*. Website <http://www.vtls.com>

Wang, J. (2002), *A Lightweight Protocol Between Visualization Tools and Digital Libraries*, Master's Thesis, Virginia Polytechnic Institute and State University.

Winer, David (1999), *XML-RPC Specification*, 16 October 1999.
<http://www.xmlrpc.com/spec>

Witten, I. H., R. J. McNab, S. J. Boddie, and D. Bainbridge (2000), "Greenstone: A Comprehensive Open-Source Digital Library Software System", in *Proceedings of Fifth ACM Conference of Digital Libraries*, San Antonio, Texas, USA, 2-7 June 2000, pp. 113-121.

Witten, Ian H., Alistair Moffat, and Timothy C. Bell (1999), *Managing Gigabytes: Compressing and Indexing Documents and Images*, Morgan Kaufmann, San Francisco.

Zeiger, Stephan (1999), *Servlet Essentials*. Available
<http://www.novocode.com/doc/servlet-essentials/>

APPENDIX A

SAMPLE XML SCHEMA FOR MDEDIT

```
<?xml version="1.0" ?>
<!--
  Annotated XML Schema for simple metadata related to an article.

  - Metadata designed for the CSTC demonstration system.
  - Annotations included for use by the MDEdit module.

  Hussein Suleman
  17 March 2002
-->

<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:jeri="http://JERICArticle"
  targetNamespace="http://JERICArticle"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

  <element name="jeri" minOccurs="1" type="jeri:jeriType">
    <annotation>
      <appinfo>
        <caption></caption>
      </appinfo>
    </annotation>
  </element>

  <complexType name="authorType">
    <sequence>

      <element name="first_name" minOccurs="1" maxOccurs="1" type="string">
        <annotation>
          <appinfo>
            <caption>First name</caption>
            <description>First name of author</description>
          </appinfo>
        </annotation>
      </element>

      <element name="last_name" minOccurs="1" maxOccurs="1" type="string">
        <annotation>
          <appinfo>
            <caption>Last name</caption>
            <description>Last name of author</description>
          </appinfo>
        </annotation>
      </element>

      <element name="email" minOccurs="1" maxOccurs="1" type="string">
        <annotation>
```

```

        <appinfo>
          <caption>Email</caption>
          <description>Email address of author</description>
        </appinfo>
      </annotation>
    </element>

    <element name="institution" minOccurs="0" maxOccurs="1" type="string">
      <annotation>
        <appinfo>
          <caption>Institution/Company</caption>
          <description>Name of the institution or company of
affiliation</description>
        </appinfo>
      </annotation>
    </element>

    <element name="department" minOccurs="0" maxOccurs="1" type="string">
      <annotation>
        <appinfo>
          <caption>Department/Center</caption>
          <description>Name of the department or center</description>
        </appinfo>
      </annotation>
    </element>

  </sequence>
</complexType>

<complexType name="jericType">
  <sequence>

    <element name="section" minOccurs="1" maxOccurs="1">
      <annotation>
        <appinfo>
          <caption>Which CSTC section are you submitting to ?</caption>
          <description>Which part of CSTC are you submitting this resource
to?</description>
          <rows>1</rows>
          <externallist>section</externallist>
        </appinfo>
      </annotation>
      <simpleType>
        <restriction base="string">
          </restriction>
        </simpleType>
      </element>

    <element name="title" minOccurs="1" maxOccurs="1" type="string">
      <annotation>
        <appinfo>
          <caption>Title of Resource</caption>
          <description>Name of the resource</description>
          <columns>40</columns>
        </appinfo>
      </annotation>
    </element>

```

```

<element name="author" minOccurs="1" maxOccurs="unbounded" type="jeri:authorType">
  <annotation>
    <appinfo>
      <caption>Author</caption>
      <description>Creator of the resource</description>
    </appinfo>
  </annotation>
</element>

<element name="description" minOccurs="0" maxOccurs="1" type="string">
  <annotation>
    <appinfo>
      <caption>Description</caption>
      <description>Summary of the resource</description>
      <rows>5</rows>
      <columns>40</columns>
    </appinfo>
  </annotation>
</element>

<element name="file" minOccurs="0" maxOccurs="unbounded">
  <annotation>
    <appinfo>
      <caption>Upload File(s)</caption>
      <description>File constituent of the resource</description>
    </appinfo>
  </annotation>
  <complexType>
    <sequence>
      <element name="url" minOccurs="1" maxOccurs="1" type="string">
        <annotation>
          <appinfo>
            <caption>Filename</caption>
            <description>Location of the file</description>
            <inputtype>file</inputtype>
          </appinfo>
        </annotation>
      </element>
      <element name="description" minOccurs="1" maxOccurs="1" type="string">
        <annotation>
          <appinfo>
            <caption>Description</caption>
            <description>Description of the file</description>
          </appinfo>
        </annotation>
      </element>
    </sequence>
  </complexType>
</element>

</sequence>
</complexType>

</schema>

```

VITA

Hussein Suleman

Education

Ph.D. in Computer Science

Date: December 2002

Topic: Open Digital Libraries

Advisor: Dr. Edward A. Fox

Virginia Polytechnic Institute and State University (Blacksburg, Virginia, USA)
(hereafter referred to as Virginia Tech)

M.Sc. in Computer Science

Date: May 1997

Topic: Genetic Programming in Mathematica

Supervisor: Dr. Miloslav Hajek

University of Durban-Westville (Durban, Kwazulu-Natal, South Africa)

B.Sc. Honours *cum laude*

Date: May 1995

Specialisation: Computer Science

University of Durban-Westville (Durban, Kwazulu-Natal, South Africa)

B.Sc. *cum laude*

Date: May 1994

Majors: Computer Science and Mathematics

University of Durban-Westville (Durban, Kwazulu-Natal, South Africa)

Selected Awards

Fulbright Scholarship

Period: August 1997 - July 1999

Awarded by US Government for graduate study in the USA

Senate Silver Medal

Date: May 1995

Awarded by University of Durban-Westville in recognition of outstanding performance as an Honours degree student

FRD Honours Programme Scholarship

Period: January-December 1994

Awarded by Foundation for Research Development's (FRD) University Development Programme to encourage post-graduate education

Faculty Bronze Medal (Faculty of Science)

Date: May 1994

Awarded by University of Durban-Westville in recognition of outstanding performance as an undergraduate student

Teaching Experience

Introduction to Data Structures and Software Engineering (CS1704)

Period: July-August 2000

Employer: Department of Computer Science at Virginia Tech

Data Structures and File Processing (CS2604)

Period: July-August 1999

Employer: Department of Computer Science at Virginia Tech

Data Structures (2nd year)

Period: February-June 1997

Employer: Department of Computer Science at University of Durban-Westville

Object Oriented Programming in C++ (3rd year)

Period: July-November 1996

Employer: Department of Computer Science at University of Durban-Westville

Assembly Language (2nd year)

Period: February-June 1996

Employer: Department of Computer Science at University of Durban-Westville

Research Experience

AmericanSouth.Org

Period: May-August 2002

Employer: Digital Library Research Laboratory at Virginia Tech, in collaboration with SOLINET and Emory University

Funding Agency: Mellon Foundation

A Digital Library of Reusable Science and Math Resources for Undergraduate Education

Period: August 2000 – May 2002, August 2002 – current

Employer: Digital Library Research Laboratory at Virginia Tech, in collaboration with Eduprise and UNC-Wilmington

Funding Agency: National Science Foundation (USA)

Web Characterization Repository

Period: May-June 1999, August 1999 – June 2000

Employer: Network Research Group at Virginia Tech, in collaboration with the W3C's Web Characterization Activity working group

Funding Agency: National Science Foundation (USA)

Papers and Publications

H. Suleman and E. A. Fox, *Designing Protocols in Support of Digital Library Componentization*, 6th European Conference on Research and Advanced Technology for Digital Libraries (ECDL2002), Rome, Italy, 16-18 September 2002.

H. Suleman and E. A. Fox, *Towards Universal Accessibility of ETDs: Building the NDLTD Union Archive*, Fifth International Symposium on Electronic Theses and Dissertations (ETD2002), Provo, Utah, USA, 30 May-1 June 2002.

H. Suleman and E. A. Fox, *A Framework for Building Open Digital Libraries*, in D-Lib Magazine 7(12), December 2001. Available <http://www.dlib.org/dlib/december01/suleman/12suleman.html>

H. Suleman, A. Atkins, M. A. Gonçalves, R. K. France, E. A. Fox, V. Chachra, M. Crowder, and J. Young, *Networked Digital Library of Theses and Dissertations: Bridging the Gaps for Global Access - Part 1: Mission and Progress*, in D-Lib Magazine 7(9), September 2001. Available <http://www.dlib.org/dlib/september01/suleman/09suleman-pt1.html>

H. Suleman, A. Atkins, M. A. Gonçalves, R. K. France, E. A. Fox, V. Chachra, M. Crowder, and J. Young, *Networked Digital Library of Theses and Dissertations: Bridging the Gaps for Global Access - Part 2: Services and Research*, in D-Lib Magazine 7(9), September 2001. Available <http://www.dlib.org/dlib/september01/suleman/09suleman-pt2.html>

H. Suleman and E. A. Fox, *The Open Archives Initiative: Realizing Simple and Effective Digital Library Interoperability*, Journal of Library Administration, 35(1/2), pp. 125-145, November 2001, Haworth Press Inc, New York. Co-published in Libraries and Electronic Resources, edited by Pamela Higgins, Haworth Press Inc, 2001.

H. Suleman, *Enforcing Interoperability with the Open Archives Initiative Repository Explorer*, Proceedings of the First ACM-IEEE Joint Conference on Digital Libraries, Roanoke, Virginia, USA, June 2001, pp. 63-64.

H. Suleman, E. A. Fox, and M. Abrams, *Building Quality into a Digital Library*, Proceedings of the Fifth ACM Conference on Digital Libraries, San Antonio, Texas, USA, June 2000, pp. 228-229.

Reviewed Conference Demonstrations

E. A. Fox, R. K. France, M. A. Gonçalves, and H. Suleman, *Building Interoperable Digital Library Services: MARIAN, Open Archives and NDLTD*, in Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New Orleans, Louisiana, USA, September 2001, p. 451.

H. Suleman, *Using the Repository Explorer to Achieve OAI Protocol Compliance*, in Proceedings of the First ACM-IEEE Joint Conference on Digital Libraries, Roanoke, Virginia, USA, June 2001, p. 459.

E. A. Fox, R. France, M. A. Gonçalves, H. Suleman, and M. H. Lee, *Building a Unified Digital Library for Theses and Dissertations*, 3rd International Conference of Asian Digital Library, Seoul, South Korea, December 2000.

H. Suleman, E. A. Fox, and M. Abrams, *Building Quality into a Digital Library*, Fifth ACM Conference on Digital Libraries, San Antonio, Texas, USA, June 2000.

Community Service

Tutorials

Building Interoperable Digital Libraries: A Practical Guide to Creating Open Archives, half-day tutorial at the Second ACM/IEEE Joint Conference on Digital Libraries, Portland, Oregon, USA, 14-18 July 2002.

Building Interoperable and Accessible ETD Collections: A Practical Guide to Creating Open Archives, Fifth International Symposium on Electronic Theses and Dissertations (ETD2002), Provo, Utah, USA, 30 May-1 June 2002.

Building Interoperable Digital Libraries: A Practical Guide to Creating Open Archives, half-day tutorial at the First ACM-IEEE Joint Conference on Digital Libraries, Roanoke, Virginia, USA, June 2001.

Workshops

Co-organised *Open Archives: Communities, Interoperability and Services*, held in conjunction with ACM SIGIR 2001, New Orleans, September 2001.

Co-chaired *Extending Interoperability of Digital Libraries: Building on the Open Archives Initiative*, held in conjunction with Fourth European Conference on Research and Advanced Technology for Digital Libraries, Lisbon, Portugal, September 2000.

Co-chaired *Extending Interoperability of Digital Libraries: Building on the Open Archives Initiative*, held in conjunction with ACM Hypertext'2000 and ACM Digital Libraries'2000, San Antonio, June 2000.

Standards Body Participation

Open Archives Initiative (OAI) Technical Committee

Period: September 2000 – June 2002

Role: Contributed to developing versions 1.0, 1.1 and 2.0 of the OAI Protocol for Metadata Harvesting, a network protocol for transferring metadata between digital libraries. Provided community support by implementing and disseminating software tools for development and testing.

Memberships/Affiliations

The Honor Society of Phi Kappa Phi

Upsilon Pi Epsilon, International Honor Society for the Computing Sciences