


Spring 2005

Lightweight Federation of Non-Cooperating Digital Libraries

Rong Shi
Old Dominion University

Follow this and additional works at: https://digitalcommons.odu.edu/computerscience_etds

 Part of the [Computer Sciences Commons](#), and the [Digital Communications and Networking Commons](#)

Recommended Citation

Shi, Rong. "Lightweight Federation of Non-Cooperating Digital Libraries" (2005). Doctor of Philosophy (PhD), dissertation, Computer Science, Old Dominion University, DOI: 10.25777/15ae-a333
https://digitalcommons.odu.edu/computerscience_etds/64

This Dissertation is brought to you for free and open access by the Computer Science at ODU Digital Commons. It has been accepted for inclusion in Computer Science Theses & Dissertations by an authorized administrator of ODU Digital Commons. For more information, please contact digitalcommons@odu.edu.

**LIGHTWEIGHT FEDERATION OF NON-COOPERATING
DIGITAL LIBRARIES**

by

Rong Shi

B.S. July 1992, Shanghai Jiao Tong University
M.S. March 1997, Shanghai Jiao Tong University
M.S. December 1999, Old Dominion University

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirement for the Degree of

DOCTOR OF PHILOSOPHY

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY

May 2005

Approved by:

Kurt Maly (Co-Director)

Mohammed Zubair (Co-Director)

Frank C. Thames (Member)

Michael L. Nelson (Member)

Johan Bollen (Member)

UMI Number: 3191378

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform 3191378

Copyright 2006 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

ABSTRACT

LIGHTWEIGHT FEDERATION OF NON-COOPERATING DIGITAL LIBRARIES

Rong Shi

Old Dominion University, 2004

Co-Director of Advisory Committee: Dr. Kurt Maly

Dr. Mohammad Zubair

This dissertation studies the challenges and issues faced in federating heterogeneous digital libraries (DLs). The objective of this research is to demonstrate the feasibility of interoperability among non-cooperating DLs by presenting a lightweight, data driven approach, or Data Centered Interoperability (DCI). We build a Lightweight Federated Digital Library (LFDL) system to provide federated search service for existing digital libraries with no prior coordination.

We describe the motivation, architecture, design and implementation of the LFDL. We develop, deploy, and evaluate key services of the federation. The major difference to existing DL interoperability approaches is one where we do not insist on cooperation among DLs, that is, they do not have to change anything in their system or processes. The underlying approach is to have a dynamic federation where digital libraries can be added (removed) to the federation in real-time. This is made possible by describing the behavior of participating DLs in an XML-based language that the federation engine understands.

The major contributions of this work are:

- This dissertation addresses the interoperability issues among non-cooperating DLs and presents a practical and efficient approach toward providing federated search service for those DLs. The DL itself remains autonomous and does not need to change its structure, data format, protocol and other internal features when it is added to the federation.
- The implementation of the LFDL is based on a lightweight, dynamic, data-centered and rule-driven architecture. To add a DL to the federation, all that is

needed is observing a DL's interaction with the user and storing the interaction specification in a human-readable and highly maintainable format. The federation engine provides the federated service based on the specification of a DL. A registration service allows dynamic DL registration, removal, or modification. No code needs to be rewritten or recompiled to add or change a DL. These notions are achieved by designing a new specification language in XML format and a powerful processing engine that enforces and implements the rules specified using the language.

- The most commonly used approach to achieve interoperability is one that harvests metadata into one central metadata repository that is then searched. One of its major drawbacks is the freshness of the data as this depends on the harvesting cycle. In this thesis we explore an alternate approach where searches are distributed to participating DLs in real time. We have addressed the performance and reliability problems associated with other distributed search approaches. This is achieved by a locally maintained metadata repository extracted from DLs, as well as an efficient caching system based on the repository.
- We also focus on service quality and usability. On the front end we introduced a dynamic user-centered, keyword driven search interface to improve service quality and usability. At the backend we provide an automatic metadata extraction mechanism to parse and process native DL search results so that the LFDL system can display rich results uniformly and consistently. A locally maintained metadata repository improves the LFDL caching system, and also makes it possible to provide additional high-level services.

As a result of our implementation work and evaluations we conclude that a federated service for non-cooperating digital libraries based on distributed search with its advantage of the freshness of data is indeed realistic, and that the dynamic, data-centered LFDL provides a lightweight and feasible approach with sufficient service quality, usability and system performance to have comparable performance of systems based on the harvesting approach.

ACKNOWLEDGMENTS

This dissertation was made possible through the support, encouragement and care of the members of my committee and many other people. First my deepest gratitude goes to my advisors, Dr. Kurt Maly and Dr. Mohammad Zubair for their guidance, patience and suggestions. Their insight, judgematic, innovative thinking guided me through the whole work and helped me overcome many of the obstacles I encountered during this work. I am also very grateful for their kindness, understanding and consideration that have help me so much both academically and personally. I wish to thank Dr. Michael Nelson, Dr. Johan Bollen, and Dr. Frank Thames, the members of my advising committee, for their careful and thorough review of this dissertation. Their insightful and incisive comments ensure the quality of this work and help me realize the importance of academic preciseness.

Many thanks are due to Imran Ameerally, who helped implementing the original prototype of this work. I would also like to thank Xiaoming Liu, Satish Kumar and other members of the digital library research group for their help and contributions.

Without the enduring support, encouragement, and love of my family I could not have gone through all those difficult times and finish this work. Thanks to my parents for their help and support during all the years. Katherine and Kevin, who were born in the midst of this work, have given me so much joy and I am indebted to them for the loss of pleasant time we could spend together. Finally I wish to give special thanks and all my love to my wife Cuc Shi for her sacrifice, understanding, and support.

TABLE OF CONTENTS

	Page
LIST OF TABLES.....	viii
LIST OF FIGURES	ix
 Section	
1. Introduction.....	1
1.1 Motivation	3
1.2 Objective	5
1.3 Approach and Issues	8
1.4 Organization of Dissertation	9
 2. Background.....	 12
2.1 DL Interoperability: Challenges and Basic Approaches	12
2.2 Distributed Search	13
2.2.1 Fully Cooperative Federation.....	14
2.2.2 Protocol Exchange	16
2.2.3 Results Gathering	19
2.3 Harvesting	20
2.4 Summary of Current Approaches	21
 3. LFDL: Approach, Architecture, and Design	 23
3.1 Introduction	23
3.2 LFDL Architecture.....	25
3.2.1 DL Specification	26
3.2.2 Registration Service	27
3.2.3 Search Service and Results Presentation Service	27
3.2.4 Management Service	28
3.2.5 Caching	28
3.3 LFDL Implementation: Rapid Prototype System	30
3.4 Discussion	32
 4. Data-centered Rules-driven Interoperability: DL Specification.....	 34
4.1 Introduction	34
4.2 Digital Library Definition Language (DLDL)	36
4.3 DL Specification Definition Using DLDL	37
4.3.1 Digital Library Content	38
4.3.2 Digital Library Access Methods	38
4.3.3 Information to be Retrieved from Digital Library	39
4.4 Discussion	41
4.4.1 DL Search Interface Capture and Query Mapping.....	42

TABLE OF CONTENTS (continued)

Section	Page
4.4.2 Search Process Simulation and Specification	43
5. Search Service: User-centered Dynamic Search	46
5.1 Introduction	46
5.2 Approach, Design, and Implementation	48
5.2.1 User-centered, Need-driven Search Mechanism	49
5.2.2 A Generic Base Universal Interface	52
5.2.3 Enhanced DLDL and DL Specification	53
5.2.4 Dynamic Interface Generation Algorithm	58
5.2.5 Additional User Customization Capability	63
5.3 Experimentation and Discussion	65
6. Results Presentation Service: Automatic Metadata Extraction	70
6.1 Introduction	70
6.2 Metadata Extraction from Non-cooperating DLs	72
6.2.1 Approach	72
6.2.2 Metadata Extraction and Parsing Process	75
6.2.3 Metadata Parsing Rules Definition	76
6.3 Experimentation and Results	78
7. Local Repository and Caching.....	84
7.1 Introduction	84
7.2 Local Metadata Repository	85
7.3 Local Metadata Search	88
7.4 Caching and Cache Replacement Algorithm	91
7.5 Experimentation, Results and Analysis	95
8. Registration Service and Management Service	98
8.1 Registration Service	98
8.1.1 Approach	98
8.1.2 Design and Implementation	99
8.2 Management Service	102
8.2.1 Real-time System Monitoring	103
8.2.2 Run-time System Reconfiguration	104
9. Conclusions and Future Work	105
9.1 Conclusions	105
9.2 Future Work	107
REFERENCES	110

TABLE OF CONTENTS (continued)

Section	Page
APPENDICES	
A. Registered Digital Libraries in the LFDL Test Bed	120
B. DTD for DLDL XML Specification	121
C. Sample DLDL Specification for ACM	124
D. Sample DLDL Specification for IEEE	127
E. Sample DLDL Specification for NEEDS.....	132
VITA.....	139

LIST OF TABLES

Table	Page
I. Comparison of Popular DL Interoperability Approaches.....	22
II. Process Specification: Other Issues	44
III. Keywords and Number of Occurrences from the DL Metadata Database	50
IV. Top Keyword-hits from Selected DLs.....	52
V. Summary of Native Form Fields Information of DLs in LFDL Test-bed.....	54
VI. Sample Query Mapping between NEEDS Native Query and LFDL UI.....	58
VII. Query Mapping to Other DLs.....	58
VIII. Search Features of Selected DLs in LFDL Federation.....	64
IX. Number of DL Native Search Results for Each Sample Query.....	67
X. Number and Accuracy of Search Results from LFDL for Each DL	67
XI. Sample DL Results and Metadata Display Patterns	74
XII. Structure of Metadata Storage Table	87
XIII. Response Time Comparisons LFDL v2 vs. v1	96
XIV. Registration Information in Memory.....	102
XV. LFDL Runtime Information	103
XVI. LFDL Registered DL Information	104

LIST OF FIGURES

Figure	Page
1.1 Architecture of a Federated Digital Library	6
2.1 Interactions of Dienst Services: UI, Index, and Repository	15
2.2 SDLIP Architecture	17
3.1 Basic LFDL Approach.....	24
3.2 LFDL Architecture	26
3.3 Caching Usage Scenarios	30
3.4 Universal Search Interface of the LFDL Rapid Prototype System.....	31
4.1 Specification Based LFDL Federation	35
4.2 Part of the DTD of a DL Specification.....	37
4.3 Specification Sample: Remote DL Access Information.....	38
4.4 Specification Sample: DL Search Interface Information.....	39
4.5 Specification Sample: Results Matching Information.....	40
4.6 Specification Sample: Multiple Results Page Information.....	40
5.1 Federated Search Service and Data Flow	47
5.2 Populating Keywords-hits for a DL.....	51
5.3 Generic Universal Search Interface	53
5.4 Native Search Interface of NEEDS	54
5.5 DLDL Schema for DL Search Field Description	56
5.6 Part of DLDL Specification for NEEDS	57
5.7 Emulated Search Interface for NEEDS Based on Specification	57

LIST OF FIGURES (continued)

Figure	Page
5.8 Dynamically Generated Search Interface for Query “html”.....	62
5.9 Dynamically Generated Search Interface for Query “university”	62
5.10 Dynamically Generated Interface when Threshold=10.....	63
5.11 Dynamically Generated Interface when Threshold=5.....	63
5.12 User Customization and Search Features Selection Interface	65
6.1 LFDL Metadata Extraction Approach.....	73
6.2 Metadata Retrieval and Parsing Workflow.....	76
6.3 Part of DTD for DL Parsing Rule Specification.....	77
6.4 Part of ACM DL Specification for Metadata Parsing	77
6.5 Part of DL Specification for Cogprints.....	78
6.6 Sample Search Results of ACM DL.....	79
6.7 Sample Results List Page and Record Page of Cogprints DL.....	80
6.8 Post Processed Results in LFDL after Metadata Parsing	81
7.1 LFDL Metadata Cache and Repository	88
7.2 LFDL Interactive Search Interface	89
7.3 Search Results Grouped by DATE.....	90
7.4 Search Results Grouped by PUBLISHER.....	91
7.5 Sample Metadata in LFDL Metadata Cache	94
8.1 LFDL Registration and Management Service	99
8.2 LDAP-based Registration Process.....	101

LIST OF FIGURES (continued)

Figure	Page
8.3 LFDL Management Service Interface.....	102
8.4 LFDL Reconfiguration Utility.....	104

SECTION 1

INTRODUCTION

Digital libraries (DLs) are the topic of research in various scientific communities. The Association of Research Libraries (ARL) indicates that there are many different definitions for digital library [6]. Generally, the computer science community may view a digital library as a networked information system with contents collected on behalf of users, while librarians may define a digital library as organizations providing services in a digital environment [12]. Essentially, a digital library is a collection of managed digital objects, comprising different types of material in different formats, which distribute across information repositories and can be accessed through wide area networks [4], [36]. Digital libraries overcome the constraints of traditional physical libraries by delivering organized, well-managed information through the Internet to anyone, anywhere, anytime. Atkins points out that “the concept of a digital library is not merely equivalent to a digitized collection with information management tools. It is rather an environment to bring together collections, services, and people in support of the full life cycle of creation, dissemination, use, and preservation of data, information, and knowledge.” [7].

Digital Libraries vs. Web Search Engines

One common question regarding DLs is “Why not just use the existing web for publishing and web search engine technology for searching published material?” It is true that although digital libraries pre-date the World Wide Web (WWW) [124], there have been major changes among DLs to adapt to the popularity and prevalence of the WWW. For example, proprietary DL search interfaces have been replaced by the ubiquitous WWW browsers, and most DLs are using the WWW-based access and transport mechanism. However, it is important to note the uniqueness of digital libraries and it is helpful to compare digital libraries with commercial web search engines such as Google, Yahoo, and Lycos, which are becoming more and more important in helping people find useful information on the web. Web search engines and digital libraries are similar in the

The journal model for this dissertation is the IEEE/ACM Transactions on Networking.

way of indexing, retrieval, storing, and searching from a user perspective. However, there are significant differences between a WWW search engine and a digital library [85]:

- The search spectrum: web search engines are aimed at the general public, which has a wide range of search requests, while digital libraries are mostly used within a specific community for education and research with users having certain predictable search patterns and behaviors.
- The contents and their management: web search engines use all the source web pages they can find on the Internet, and they have no control or intrinsic management over the distributed pages. The contents of a digital library are well defined and well organized, and specifically, objects have metadata associated with them. A DL provides acquisition, management, and maintenance processes to manipulate the digital objects [85].
- The user interface and the service: the interface of web search engines is fairly simple, typically in the basic mode containing only a keyword field. The quantity of search results and the response time of a particular search are more important than search quality. As for digital libraries, however, service quality or search accuracy is the most important factor. There are more search fields, based on the metadata associated with the digital objects in the DL, to filter out unnecessary information, enabling a more accurate result.

Considering the above three factors, digital libraries differ from web search engines in their internal structure and implementation, from indexing and archiving to search algorithms. General web search-engines have solved the interoperability problem by developing sophisticated crawlers but have significant problems with obtaining results from the “hidden” web [9] that digital libraries inhabit. Also these engines have no way to take advantage of metadata that may be available to characterize web pages (though there are some recent efforts with the semantic web [11], [50] and RDF [102]). However, when building service on top of existing, distributed DLs, as we will discuss in detail throughout this dissertation, we can use or adapt approaches and methods derived from the experience of building web search engines.

1.1 MOTIVATION

DLs are now commonly used in science, technology, engineering and the arts. A number of successful digital libraries have been built to manage and disseminate collections of information beyond the scope of traditional libraries. Some examples are described in NSDL (National Science, Mathematics, Engineering, and Technology Digital Library) [56], [90], [91] and DLI-2 (Digital Libraries Initiative phase 2) [26], [37]. However, like traditional libraries, each organization is responsible for its own DL implementation and most of the libraries have been built in isolation utilizing different technologies and protocols. Each library has its own publication and search interfaces, its own interpretation of metadata formats in terms of both syntax and semantics, and its own management policy.

This uncoordinated development approach was adequate in the early stages of DL and WWW technology, but DL technology is currently included in the strategic planning of many institutions. The differences in DL implementation hinder the development of digital library services which enables users to discover information from multiple libraries through a single unified interface. To build an effective information infrastructure that can meet the growing demand, it is necessary to integrate heterogeneous information resources and build interoperable services.

The ideal approach to interoperability is to have all DLs use the same software or common protocol. However, that is unrealistic and there are enough significant DL systems in use to assume that the DL community will continue to support a number of heterogeneous systems and protocols [133]. It seems likely that over time, a handful of DL protocols and systems will have sufficient functionality and installed base, preventing a convergence to a single system. Therefore, digital library interoperability is an active research field in the DL community. Andreas Paepcke describes interoperability as cooperating systems where individual components are designed or operated autonomously [96]. He suggests that “the ultimate goal for such a system is to have components evolve independently, yet to allow all components to call on each other efficiently and conveniently.” [96]

From a technical point of view, there are basically two approaches to build interoperable service across individually independent digital libraries: a metadata

harvesting approach [13], [14] and a distributed search approach [105]. The former would require data providers (those maintaining individual repositories) to expose metadata by following a common metadata harvesting protocol so that an end-user service provider (those providing search or other services) can utilize the harvested metadata to provide search service and other high-level services. In a distributed search, a service provider will distribute the user query to each individual DL in real time, and collect results from them, and then present users with the merged results.

Both approaches achieve the goal of interoperability by providing high-level federated end-user search service while making it possible for each individual digital library to operate independently. Harvesting can provide scalable, robust search services and various value-added services on the collected metadata, but typically it requires an archive to implement the harvesting protocol and to expose its metadata. Alternatively, a crawler harvesting approach does not require a protocol or expose metadata but it only works best for non-structured or semi-structured data and also has a data synchronization problem. The search results from harvested data may be not as fresh as the ones from a DL if a user accesses that DL directly. The distributed search may or may not require of implementing a joint protocol or agreement, and its search results maybe fresher or more close to what a user can get from a DL directly. However, the distributed search has important problems of system performance, reliability, and scalability.

Currently there are a number of research projects on DL interoperability being conducted by leading research organizations and universities. We have conducted a thorough survey and study of those projects, which follow either distributed search or harvesting approach. One of the issues or limitations is the cost to participate in the interoperation. The burden is either on the participating DL side, or on the service provider side. Either way, significant effort is required in each DL, like using a new protocol, changing data format, and installing a new software suite; or great effort is required in the management of the interoperability system. Whenever a new DL is added, or an existing DL changes its behavior, the whole interoperability system needs to be changed, e.g., adding new code, changing existing code or existing interface, recompiling and restarting the system.

We do not want to simply assert that distributed search is superior over harvesting, or vice versa. We think there are enough digital libraries that want to be autonomous and for various reasons do not want to make any effort, such as adopting a joint protocol or exposing metadata, to participate in an interoperation. We are especially interested in the interoperability among those totally non-cooperating DLs. In this case, a pure harvesting approach is not possible but it is feasible to provide a federation based on distributed search. Therefore, in this dissertation we concentrate on building interoperable service across heterogeneous sources using fundamentally the distributed search approach. However, we also study the possibility of utilizing the features of harvesting to address the issues of the distributed search so that the service built can take advantage of the both: a scalable, robust search service with fresh results.

We believe it is crucial that such a federated search service, based on a combination of distributed search and harvesting, should be flexible and lightweight, both easy to use by end users and easy to manage by an interoperability service provider (while at the same time preserving each individual DL's autonomy).

The challenges to such lightweight approach are:

- The integration should be flexible enough to allow individual participants of the federation to add/modify features and at the same time maintain the user's impression of a single system.
- Relocation, addition, deletion of individual DLs should be transparent to users.

The service should not depend upon, or even care about, the implementation of any particular search service. The underlying architecture of the individual digital library should be unimportant. As long as individual search services are openly accessible, a lightweight, distributed search approach can provide the benefit of accessing them simultaneously and collating the results.

1.2 OBJECTIVE

The objective of this research is to demonstrate the feasibility of interoperability among non-cooperating digital libraries by building a federated digital library. The federation shall be dynamic, flexible, and lightweight. Existing DLs can remain autonomous and do not have to change anything in their system or processes. It shall be

easy to add DLs to the federation and the newly added DLs shall be incorporated into the service in real time: no code needs to be rewritten or recompiled. We aim for system usability, feasibility and applicability to various domains, performance, and service quality in our approach. To achieve this objective we develop, deploy, and evaluate key services of the federation. The fundamental underlying approach is to distribute searches across DLs without prior coordination, but we also want to explore the feasibility of taking advantage of the harvesting approach to address performance and scalability issues related to distributed search.

Figure 1.1 illustrates the layered architecture of our proposed federation and its services; the goals for each service are summarized below.

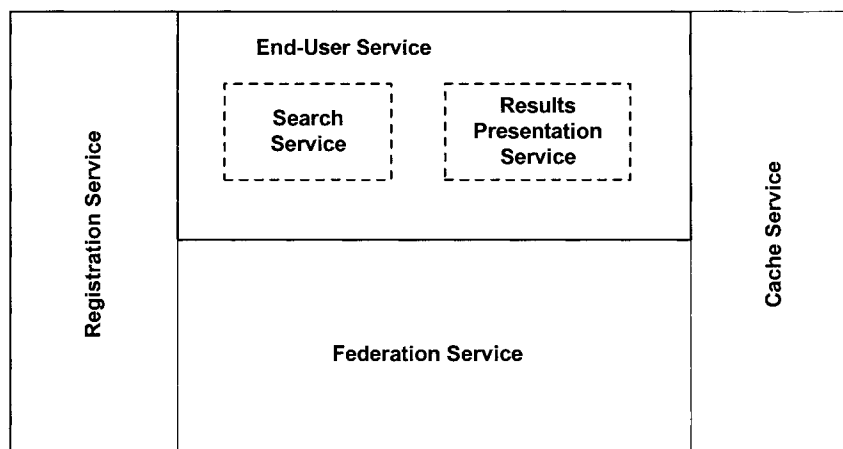


Fig. 1.1. Architecture of a federated digital library.

Federation Service The *federation service* is the key service, which incorporates numbers of non-cooperating digital libraries to form a federated library. Since the DLs lack cooperation, we study methods of collecting a DL's interoperability information by observing its external behavior. For each digital library a specification describes the rules to be used by the federation service on how to send out distributed query and collect search results. Our objective is to design a flexible, easy to understand and implement, universal schema. We expect to achieve a well-managed information system that has performance and efficiency equivalent to that of the

harvesting approach. The *management service* is part of the federation service which facilitates the monitoring and maintenance tasks of the federation system. It collects various system statistics data for troubleshooting and possible system enhancement. The management service also allows for fine-tuning the system to ensure a service with the best performance and reliability.

End-User Service Once specifications about DLs are available and a federation of heterogeneous digital libraries is formed, various services can be built for end users. Search users expect fresh, accurate, results from multiple sources accessing a single, easy to use search interface. Our objective is to develop a *unified search service* that works with the federation service to distribute the queries to all underlying DLs. The *results presentation service* shall collect and process results from different DLs and then present the merged results to end users in a consistent way as if all results are coming from one single source. Quality and usability will be the critical metrics for end users' satisfaction.

Registration Service To provide the federation service among distributed, autonomous digital libraries, the service provider needs to be aware of the existence of a DL repository. The *registration service* allows a new DL to be added to the federation by registering its specification. Our objective is to make the process dynamic and transparent to end users: to add a new DL, no code change should be necessary and the newly joined DL shall be part of the federated search on the fly.

Cache Service The *cache service* optimizes the system performance and reliability of the federation by caching most recently used search results. Our objective is to alleviate one of the significant issues with the distributed search, namely, response time to get all results presented to the user. We design an intelligent caching schema that will extract metadata from previous searches to help in using even partial answers to queries that can be provided to the search user immediately.

1.3 APPROACH AND ISSUES

In this dissertation we investigate a lightweight and general approach to interoperability - Data Centered Interoperability (DCI) and build a federated search service for DLs without prior coordination among the participating DLs. The outcome of our work will be an operational version that will on the one hand demonstrate the feasibility of our approach and on the other hand allow us to study various issues with the distributed search approach.

- The feasibility of interoperability of non-cooperating DLs

In our approach we observe and capture users' interaction with a DL and build a federated service based on all possible user/DL interactions including the way a DL presents the results of a query to the user. We propose to study a DL's external behavior without the knowledge of its internal structure and implementation. Therefore, existing DLs can continue their operation without having to add code or expose their objects to the federation beyond what the DL does for its own community. A DL may change its externally observable behavior from time to time. Therefore, our solution will have to have the ability to discover change and then have mechanisms to adjust accordingly.

- The architecture of building a federation service

In our approach we propose a data-centered and rules-driven architecture, that is, the core engine should not depend in any way on a particular DL's behavior. Instead, the code should use specifications of DLs' behaviors and have rules on how to interpret them. We propose to design a standard XML [125] based DL metadata sheet to describe each DL's specification: its characteristics, capabilities, and interoperability information. All tasks should be performed by the federation system according to the rules defined in each DL's specification from query mapping to results processing.

Some DLs have complex search interfaces or require comprehensive user interactions, which may be difficult to capture using an XML specification.

In our approach, each DL registers with a central registration service provider. DL registration, removal, and modification should be dynamic, easy in management and maintenance, and transparent to end-users. We want the system

such that when it is operational and running, a new DL can be added and an existing DL can change its behavior or be removed on the fly. The issue is how to achieve this without any code change or system restart.

- Service quality and usability

The objective of the search service is to provide universal search interface through which the user is presented with accurate results promptly. The issue is how to present the user with a dynamic interface that depends on the user, her preferences, her past queries, and her input based on the profile of the target DLs, as well as the user' needs.

- System performance and robustness

The distributed search approach provides fresh results while suffering response time and reliability issues. We propose mechanisms that will automatically discover and store metadata from queries and their results. To address the issues of system performance, availability and robustness, we will design a local metadata repository architecture with caching and a pre-fetch mechanism. This approach though raises another issue: Why not just use harvesting instead? A simple answer is that there are DLs that will not actively participate in making their metadata available. Therefore, the only alternative is for the service provider to discover and retrieve metadata on demand. The more complex answer is that we do not know yet the actual tradeoffs involved in the two approaches without having operational systems to evaluate them.

1.4 ORGANIZATION OF DISSERTATION

In this dissertation we present the issues and challenges during the design, development, implementation, deployment, and evaluation of the federated digital library system and then describe our experimental solutions to those issues. The rest of dissertation is organized as follows:

Section 2: Background

We introduce various DL interoperation approaches. For each approach we present typical systems, their advantages and disadvantages, and we summarize a comparison of those approaches.

Section 3: LFDL: Approach, Architecture, and Design

We present our approach to achieving interoperation among non-cooperating digital libraries in this chapter, and the overall architecture of the federated system to provide federated service by implementing the approach. We describe our experiences in building a prototype system and discuss limitations and issues such as quality of service, search usefulness and usability, along with system robustness and performance.

Section 4: Data-centered Rules-driven Interoperability: DL Specification

The key to a lightweight, flexible federated service is a DL's specification which describes a DL's characteristics and features. In this chapter a digital library definition language is introduced to specify DL's interoperability information.

Section 5: Search Service: User-centered Dynamic Search

Section 5 addresses the service quality and usability issue of the federation by providing a user-centered, need-driven, interactive search mechanism. We use Dublin Core as the basic interoperation middle layer, and a dynamic query mapping mechanism to map between the common layer and the native libraries' layers.

Section 6: Results Presentation Service: Automatic Metadata Retrieval and Harvesting

Results processing is another characteristic that distinguishes our system from other distributed search approaches. The federated DL can display search results from different DLs in a consistent way so the end users are unaware of the different presentation mechanisms used by the participating DLs. Organizing the result set helps a user to locate the target object quickly. This requires post-processing of the result set using all the metadata available from the result set, which is a difficult task in the distributed query

approach. In this chapter we present an automatic metadata discovery and retrieval mechanism by observing the external behavior of a DL. The digital library definition language has been enhanced and the XML specification of a DL is used to define the rules to obtain metadata from each DL's result pages.

Section 7: Local Repository and Caching

Section 7 describes how the federated digital library uses the retrieved metadata to build a local metadata repository. Based on the local repository we design and implement an intelligent cache to improve the performance and robustness of the federated service. We also use a secondary level in-memory cache to further improve the system efficiency.

Section 8: Registration Service

Section 8 gives details on the design and implementation of the registration service for the federated DL.

Section 9: Conclusions and Future Work

Section 9 summarizes our work on DL federation with the major contributions highlighted, as well as the major issues we addressed and those we have not. We provide directions for future work on those unresolved issues.

SECTION 2

BACKGROUND

The DL federation addresses the DL interoperability by building a coherent set of digital library services that enables users to find information from multiple sources through a single unified interface [67]. In this chapter we discuss previous work in this area. This chapter is organized as follows:

- We begin with an introduction to the challenges and basic approaches to DL interoperability.
- We then discuss the distributed search approach in section 2.2. We present some typical systems, their advantages and disadvantages.
- Next, in section 2.3, we describe the harvesting approach.
- Finally, in section 2.4, we have a summarized comparison of the approaches discussed and where the LFDL fits in.

2.1 DL INTEROPERABILITY: CHALLENGES AND BASIC APPROACHES

Digital libraries are important tools and being used in many scientific and technical disciplines. However, as mentioned in Section 1, most of these DLs are implemented using protocols specific to the field they support and much work has to be done to achieve interoperability among DLs on a large scale [133].

To end-users interoperability of digital libraries means a seamless presentation of a federation of DLs. As identified by the NSDL community, DL interoperability can be achieved at three levels: technical, content and organizational:

“Technical agreements cover formats, protocols, and security systems so that messages can be exchanged, etc. Content agreements cover the data and metadata, and include semantic agreements on the interpretation of the messages. Organizational agreements cover the ground rules for access, for changing collections and services, payment, authentication, etc.” [91]

In this dissertation, we focus on DL interoperability at technical level.

The approaches for technical interoperability can be categorized into two basic types, distributed search and harvesting. In the distributed search approach a unified search service provider distributes the search query to multiple standalone DLs simultaneously and then either processes the results from each DL and presents the results in a consistent manner, or just simply returns results as each DL's native format without any processing. Query results processing maintains transparency to the end-users of the underlying DLs as well as provides other high-level services. In the harvesting approach a service provider collects metadata from heterogeneous sources and then provides search service based on the metadata harvested.

Generally speaking, the distributed search approach may provide more accurate¹ and fresher search results but may require implementation of a joint distributed search protocol. Moreover, it may suffer performance, reliability and scalability issues. On the other hand, the harvesting approach has better scalability and can provide enhanced services based on harvested metadata; however, it also has the issues of repository synchronization for maintaining freshness. Still yet, it requires participants to adopt a harvesting protocol while some DLs may not be able or willing to do so.

2.2 DISTRIBUTED SEARCH

In a distributed search queries are sent out to each DL. Subsequently search results are retrieved, merged and presented to users. There are three typical distributed search models: 1) a fully cooperative federation in which participants adopt the same software; 2) a protocol exchange and interoperation in which DLs follow the same protocol agreement; 3) a results gathering approach in which no effort is required from individual DL but the service provider is totally responsible for gathering information from each DL to provide a federated search service [67].

The first model provides the most complete form of interoperability, but requires great efforts from its participants. At the other end results gathering requires little from participants, but to provide the same quality of service as one that provided by a fully cooperative federation, extra work needs to be done by the interoperability service

¹ "more accurate" here refers to that the results from the distributed search more closely represent the results that a user can get from a DL by accessing it directly.

provider. Below we will describe each model in detail and then give some sample solutions of that model.

2.2.1 Fully Cooperative Federation

In a fully cooperative digital library system all participants use the same DL protocol and software implementation which means all organizations have to use the same computer systems or software suite. Currently a fully cooperative model has been somehow obsolete as of the inflexibility it imposed on the participants. However, it has played an important role in the evolution of DL interoperability and many lessons can be learned such as user interface design and DL registration service. Some systems like NCSTRL have evolved and adopted new model of interoperation.

NCSTRL/DIENST

NCSTRL, or Networked Computer Science Technical Reference Library, is a confederation of over 100 institutions with the goal of providing a federated search service centered on computer science material [22], [28], [39]. Each organization maintains its own digital library services and the interoperability is achieved by conformance to an open architecture and joint protocol, agreement on data types and metadata format [64].

Dienst is the protocol used in NCSTRL [23], [55]. It specifies an open extensible protocol for the interoperation among various digital library services so that resources can be accessed universally [22]. Dienst consists of 5 components: 1) Repository Service; 2) Index Service; 3) Meta-Service; 4) User Interface (UI) Service; and 5) Library Management Service. Figure 2.1 shows some of the Dienst services and their interactions. The UI service communicates with end users using the standard HTTP [31] and HTML, and with other services such as Index and Repository service using the Dienst protocol. Sample NCSTRL/Dienst based digital libraries can be found at [75], [86], [131].

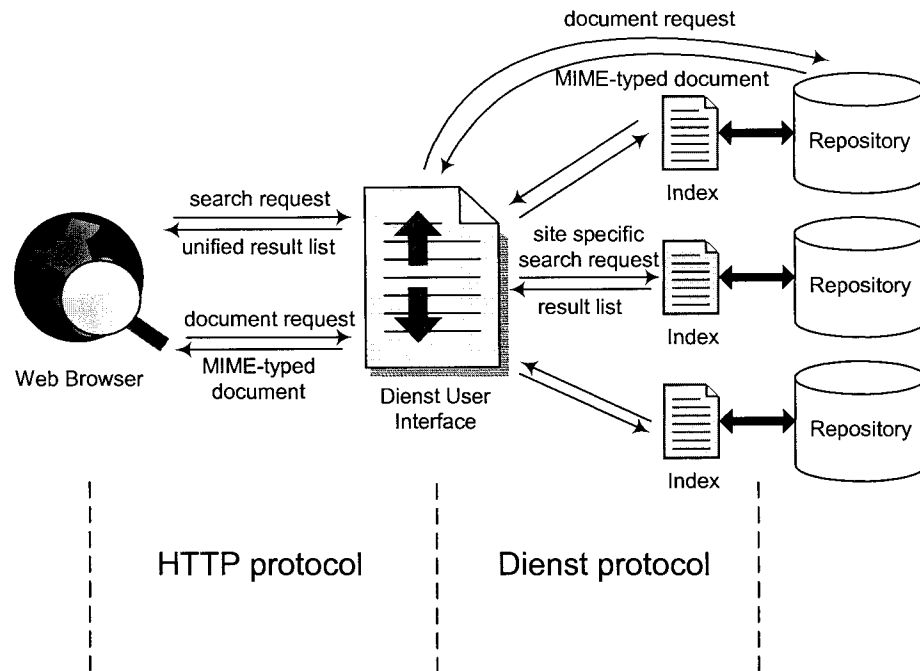


Fig. 2.1. Interactions of Dienst services: UI, Index, and Repository.

NCSTRL was popular in the online publishing of computer science technical reports among colleges. However, the drawback of this approach is also becoming obvious: whenever a new DL wants to join this federation, it has to install the standard software package that implements Dienst protocol, then coordinate with the federation service provider (like www.ncstrl.org) to add itself to the federation; whenever there is a software version update or other code change, participants have to get the new version and run it again. Although an organization may want to implement some add-on features which are most suitable for its own data or structure, there is no way to do it unless the standard protocol and software implements those features. NCSTRL also suffers reliability and scalability problems [100].

NCSTRL was originally developed and maintained by Cornell University until 2001. Due to the problems mentioned above, since 2001 NCSTRL has been migrated to an OAI-PMH based architecture [2].

Obviously the cost of participation a fully cooperative federation is high as DLs have to implement and keep current with all the protocols and agreements. There are a lot more significant autonomous DL systems and there is no doubt that the current

heterogeneous systems and protocols will continue to evolve other than disappear. Therefore, fully cooperative federation is a far less feasible solution to DL interoperability.

2.2.2 Protocol Exchange

In this model the distributed search is achieved by each participant implementing protocol agreement on information exchange among DL search services. Some well-known standards and protocols are Z39.50, STARTS, SDLIP, and GINF.

Z39.50

Z39.50 is an international standard for communication among information systems [129]. It specifies the protocol on information searching and retrieval from different computer systems independent of the internal structure of each information resources [44]. Gateway to Library Catalogs is a web-based search interface to search the Library of Congress catalog as well as hundreds of other institutions utilizing the Z39.50 protocol [65]. Once popular but now obsolete, WAIS or Wide Area Information Servers [48] is a distributed text searching system that uses the Z39.50 to search indexed text-based information system across wide area networks. The Z39.50 is a comprehensive standard, but is often too large and complex to be applied to light-weight, open source systems typical in web-based solutions and applications. To implement a Z39.50-based system the flexibility and options offered by Z39.50 can be overwhelming. The interoperability could be compromised if a client implements some features but a server supports some other features [127].

STARTS

STARTS [8], [34], the Stanford Protocol for Internet Retrieval and Search is a protocol for information retrieval from multiple collections of text documents developed by Stanford University and over 10 other organizations. The goal of STARTS is to develop *metasearchers* [34] that can discover the most suitable sources for a given query, retrieve, evaluate and then merge results from those sources. One of the issues associated with STARTS is that it does not cope with complex searches for non-document objects

[127]. Also, STARTS depends on simple but expressive agreement among service providers to achieve interoperability. For example, it requires that each information source describes itself by exporting its general metadata information. This may not be possible as some providers have proprietary internal structure that they are not willing to reveal.

SDLIP

SDLIP, the Simple Digital Library Interoperability Protocol, is a middleware approach to achieve interoperability developed by Stanford University [95]. In SDLIP a wrapper or digital library proxy is defined between the search client and the ultimate information source. Between the client and the proxy SDLIP defines the transport protocol, query language, and other interface so that they can communicate. Clients use SDLIP to request searches to be performed over information sources. The transport protocol can be HTTP or CORBA [19] based.

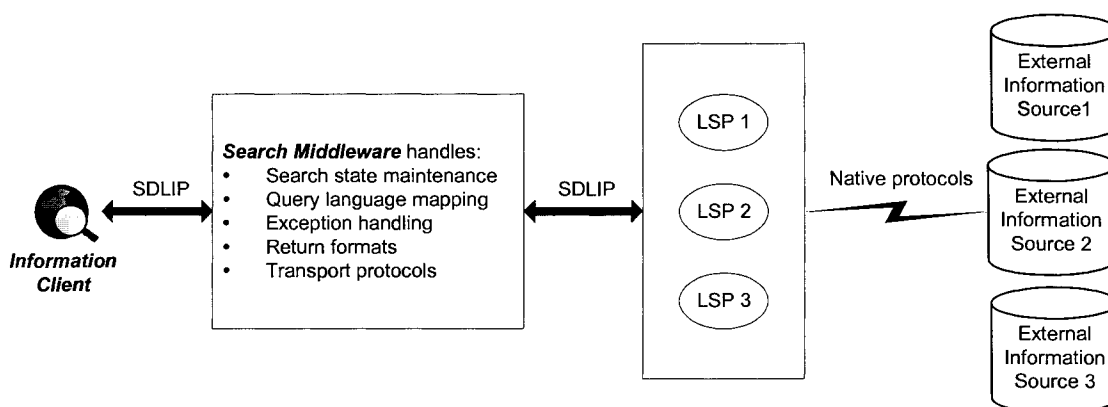


Fig. 2.2. SDLIP architecture.

As illustrated in Figure 2.2, an ultimate external information resource may or may not implement SDLIP directly. A Library Service Proxy (LSP) can wrap multiple external sources and communicate with them via native protocols required for these services. At the front end, SDLIP mandates the interaction between LSP and information client. A client can also access a resource directly if the resource is already SDLIP compliant.

One of the problems with SDLIP is the proxy approach: for each DL a separate proxy code is needed. This is not efficient: each time a new DL is added, or a registered DL changes its behavior, the proxy code for that DL has to be changed and recompiled. Another problem is that on the client side a protocol library or API is needed and installed. Though Java Applet can be utilized, most of the users still prefer the standard and efficient web interface. It is true that this protocol allows the client to be applications or devices other than a web browser, but to the digital library community, a thin client using web browsers and pure HTML forms is enough, and more efficient. Also in SDLIP, though there is a universal search interface, users still have to send request to each DL one by one, but cannot use one interface to send a request once to query all the DLs they want to search.

GINF

The goal of Generic Interoperability Framework (GINF) is to achieve interoperability across heterogeneous information resources which have various protocols, query languages, and data formats by providing a uniform interface to access those sources [79]. It attempts to develop a generic framework to universally represent different protocols, languages, data and interface descriptions while at the same time preserving their semantic variety. The current implementation uses RDF [102] to define all protocols and formats.

GINF is more generic than SDLIP, but essentially they are based on the same approach. GINF is working on modeling the SDLIP protocol using its own protocol model. Like SDLIP, though there maybe less burden for each information source it tried to integrate, significant work is needed on the GINF system itself. Whenever a new DL is to be incorporated, a lot has to be done to define the DL's structure, data and protocol using the GINF protocol. More generic does not signify simpler and easier. Additional work needs to be done to achieve a more generic style; sometimes generic also means that you have to suffer with performance issues and other maintenance problems.

SDARTS

SDARTS is a protocol and toolkit designed at the Computer Science Department of Columbia University to combine two complementary existing protocols, SDLIP and STARTS [35], [45]. SDARTS is essentially an instantiation of SDLIP, but with added elements from STARTS. Specifically the specification on the metadata a resource should export to facilitate meta-search among multiple sources.

SDARTS makes it possible to build interoperable search service among non-cooperating web-based digital libraries. However, it is built upon two protocols, SDLIP and STARTS, which means layered architecture with both clients and servers have to be developed according to the standards. This may not be a really lightweight approach. A registration service is needed and the result parsing is limited. Writing collection configuration files requires thorough knowledge of STARTS and XSLT, which may be easier for a programmer but not for DL experts.

2.2.3 Results Gathering

It is still possible to achieve interoperability among DLs that are not prepared to cooperate in any formal manner. This can be done by gathering openly accessible information, from search interface to search results [91]. The results gathering approach uses the distributed search approach and it does not require any prior coordination among federated digital libraries. We think this is a common scenario, and our approach in the LFDL falls into this model. The most common examples of this approach are the Web search engines. Because there is no cost to participate, it is possible for results gathering to provide services that embrace large numbers of digital libraries; however, if there is no extra work to control the quality, the services are usually of poorer quality than can be achieved by partners who cooperate more fully.

Commercial meta web search engines

Strictly speaking, the popularly used commercial meta WWW search service, or a search engine of search engines, like MetaCrawler [80] and search.com [106], are not for interoperable digital libraries. But like commercial web search engines, though they are much different than digital libraries, technically, there are numbers of similarities.

MetaCrawler: Currently dozen of search services are available ranging from general purpose to special need. Each service has its unique interface and may only return partial or irrelevant search results. To get comprehensive, useful information users may have to use different services for the same query and also manually find out useful results. The MetaCrawler provides a single, universal interface for Web search. It distributes a query to multiple search engines in parallel, then processes the results, and finally returns those validated, relevant results to users [108].

Commercial meta web search engines like MetaCrawler are essentially using the gathering approach to provide a meta search service. Determined by their nature search result quantity and response time are always the top priorities, while the quality of service is not as important. However, to the digital library community, quality of service is always most important.

SearchLight

SearchLight is part of the California Digital Library initiatives and its goal is to search multiple public databases and other information resources at one time [107]. Currently Searchlight has integrated quite a few digital libraries. However, to achieve uniformity across resources, it is relatively generic without post processing search results. To get more precise results one may have to search a resource directly. SearchLight also has performance and reliability issues as it depends on the real-time response from each source.

2.3 HARVESTING

The harvesting approach for digital library interoperability is to collect metadata from heterogeneous sources to form one homogeneous collection [67]. Formerly called UPS (Universal Preprint Service), the Open Archives Initiative (OAI) [58], [93] is based upon the concept of metadata harvesting. The OAI defines the format of metadata each digital library should expose as well as the protocol on how to retrieve metadata. The underlying type of content of each library and the internal structure of its service are irrelevant.

The core of the OAI is the metadata harvesting protocol, OAI-PMH (Open Archives Initiatives Protocol for Metadata Harvesting) [57], which specifies how to transfer

metadata from a data provider to a service provider. It contains the following service requests or verbs: Identify, GetRecord, ListIdentifiers, ListRecords, ListSets, and ListMetadataFormats.

Arc is the first federated searching service based on the OAI protocol [69], [70]. Numbers of other DL applications are OAI based, such as Kepler [68], [76], Archon [77], and DP9 [71] which are the research projects that are being conducted by the Digital Library Group of the Old Dominion University [25].

The burden of participating in a harvesting based federation is much less than that of participating in a fully cooperative federation, therefore more organizations may be able to join a federation by harvesting while still keeping their existing systems. However, though the efforts required for participants are less, they still need to adopt certain agreement, and currently there are significant numbers of autonomous DLs that either not willing to or not able to adopt outside standards. Also a service provider has to be aware of the data freshness and data synchronization issues.

2.4 SUMMARY OF CURRENT APPROACHES

In Table I we give a technical summary of the major approaches we have discussed. In comparison we have also included here the LFDL approach. Out of the approaches mentioned only the OAI utilizes the harvesting approach and we are mostly interested in the distributed search because of our focus on existing non-cooperating digital libraries. NCSTRL/Dienst are somewhat obsolete, and among other distributed search approaches all of them provide a unified search interface and have query translation between a universal interface and a native DL interface. SDLIP and GINF are layered, protocol based approaches. They define the underlying communication protocol which can be TCP or CORBA. As a results gathering methodology, SearchLight uses the high level communication protocol, HTTP. Though not much work on the data provider side, most of the current distributed approaches somehow require great effort from service providers, either to implement common protocols or to write separate code for each DL incorporated. One of the design goals of the LFDL is for it to be lightweight to the service provider also.

TABLE I
COMPARISON OF POPULAR DL INTEROPERABILITY APPROACHES

	NCSTRL	OAI	Meta web search engine	SDLIP SDARTS	GINF	SearchLight	LFDL
Basic approach	Distributed Search: Federation	Harvesting	Distributed Search: Gathering	Distributed Search: protocol	Distributed Search: protocol	Distributed Search: Gathering	Distributed Search: Gathering
Universal search interface	Yes	N/A	Yes	Yes	Yes	Yes	Yes
Simultaneous search multiple DLs	Yes	N/A	Yes	No	No	Yes	Yes
Response time=worst DL?	Yes	N/A	Yes	N/A	N/A	Yes	No
Allow asynchronous search	No	N/A	Maybe	Yes	No	No	Yes
Publish	Yes	No	No	No	No	No	No
Query/Protocol translation	No	No	Yes	Yes	Yes	Yes	Yes
Transport protocol	HTTP	HTTP	HTTP	TCP, Corba, HTTP	TCP, HTTP	HTTP	HTTP
Metadata format	Any	XML	N/A	Any	RDF	N/A	XML
Cost	High: data providers Low: service providers	Median: data and service providers	None: data providers High: service providers	None: data providers High: service providers	None/high: data providers High: service providers	None: data providers High: service providers	None: data providers Low: service providers

Basically the LFDL falls in the results gathering model of the distributed search approach. However, it also integrates some sort of harvesting approach by utilizing a locally maintained repository of metadata extracted from remote DLs. Details about the LFDL will be given in later sections.

SECTION 3

LFDL: APPROACH, ARCHITECTURE AND DESIGN

In the previous chapter, we discussed various approaches to digital library interoperability. The distributed search and results gathering is one way to implement an interoperable federation system. The LFDL we propose is designed to provide a unified, federated search service for heterogeneous, non-cooperating collections. This chapter presents the basic approach, design goals and architecture of the LFDL.

The remainder of this chapter is organized as follows:

- Section 3.1 introduces the basic approach taken by the LFDL.
- Section 3.2 describes the design goals and the services planned for the LFDL. We then define the overall architecture of the LFDL.
- We review the effort of designing and implementing the initial LFDL prototype system in section 3.3.
- Finally in section 3.4, we analyze the experiences, issues, and lessons of the building of the LFDL system and discuss related work.

3.1 INTRODUCTION

One major objective of interoperability among existing independent, non-cooperating digital libraries is to provide a federated service with a unified search interface, so that users can utilize the interface to seamlessly search across multiple repositories simultaneously [67]. The distributed search and results gathering represents a straightforward approach. The LFDL follows this approach and provides a federated search service for end users.

As illustrated in Figure 3.1, in the LFDL, a query submitted from the User Interface by a user is translated to a native format of a particular digital library, and the native query is sent to the corresponding DL. Once search results are received from various sources, they are merged and presented to the user. Thus, an integrated service is provided and yet users are unaware of the underlying heterogeneous information providers.

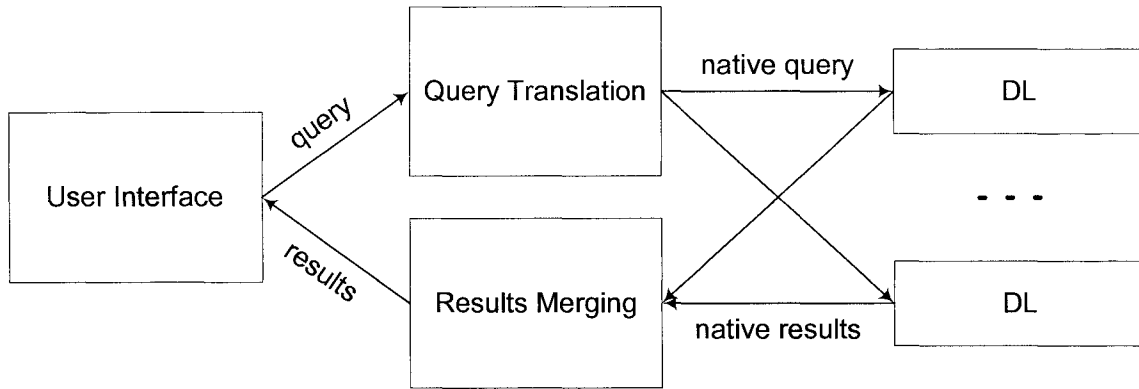


Fig. 3.1. Basic LFDL approach.

In Section 2 we discussed and compared several models following the distributed search approach. One of the issues in the basic distributed approach is that though it alleviates the burden on data providers to join a federated system, a great effort is required from service providers to include a new data source to the system. For example, one has to write new code specific to the new source and add it to the current package, and then redeploy the whole package. Moreover, whenever there is any change within any data source, from search interface to results presentation, the same process has to be repeated.

One of the design goals of the LFDL is that it shall be a flexible, lightweight solution, both to data providers and the provider of the federated service: to data providers, the LFDL is a non-issue as no extra work is required. Therefore existing DLs' structure or protocol can be kept intact while being in the LFDL federation; to the service provider, which is the LFDL system itself, it should be a small effort. Once the system is started, little effort should be needed to keep it running, no new code to install to add a new DL, no recompile or restart of the system.

We present a data-centered and rules-driven approach to achieve the design goals [132], [133]. The key is to create a specification to describe the behavior of each newly added DL source. The specification defines the rules of query mapping between the LFDL query and the native query of a DL, as well as the rules on how to interpret and process search results from a DL. By enforcing the rules the LFDL can perform a federated search against multiple sources and present the merged results. Ideally, the

experts from each individual DL are the best people to create those specifications. We aim to design the LFDL to be flexible enough so that after a short period of practicing, anyone with a basic understanding of our approach can integrate a new DL into the LFDL system. And once the DL has been added, end users should be able to search it using a universal interface without any delay.

3.2 LFDL ARCHITECTURE

As stated in Section 1, the design goal of the LFDL is for it to be a lightweight, flexible approach based on robust and efficient architecture, to achieve a federated service with adequate service quality, usability and performance.

Figure 3.2 shows the services and major components of the LFDL we propose. The core service is the Search Service and Results Presentation Service for end users. In addition, as a federation of distributed information sources, a Registration Service is necessary to reveal where resources are located and what capabilities these DLs have. A Management Service enables the administrator to monitor and fine-tune the LFDL to achieve better system efficiency and performance.

The LFDL services are implemented by the LFDL Federation Engine, which consists of a number of sub modules. On the front end end-users employ the Universal Search Interface to access the LFDL federated search. At the back end each participating DL (or rather the person responsible for the addition of this to come from the DL's parent organization) registers its specification with the LFDL to describe how to access its library. Once a user submits a search request through the universal search interface, the LFDL search service will use the query mapping rules from a DL specification to translate the query to that DL's native query. The LFDL will then send the translated query to a remote DL and get results back. The LFDL results presentation service will parse the result set and save it to Cache and then display it to end users.

DL experts can access the LFDL Registration Service interface to add a digital library and its specification to the federation. The LFDL system manager can utilize the Management Service interface to conduct system monitoring and maintenance tasks.

In the next sections we shall give details on the major LFDL components.

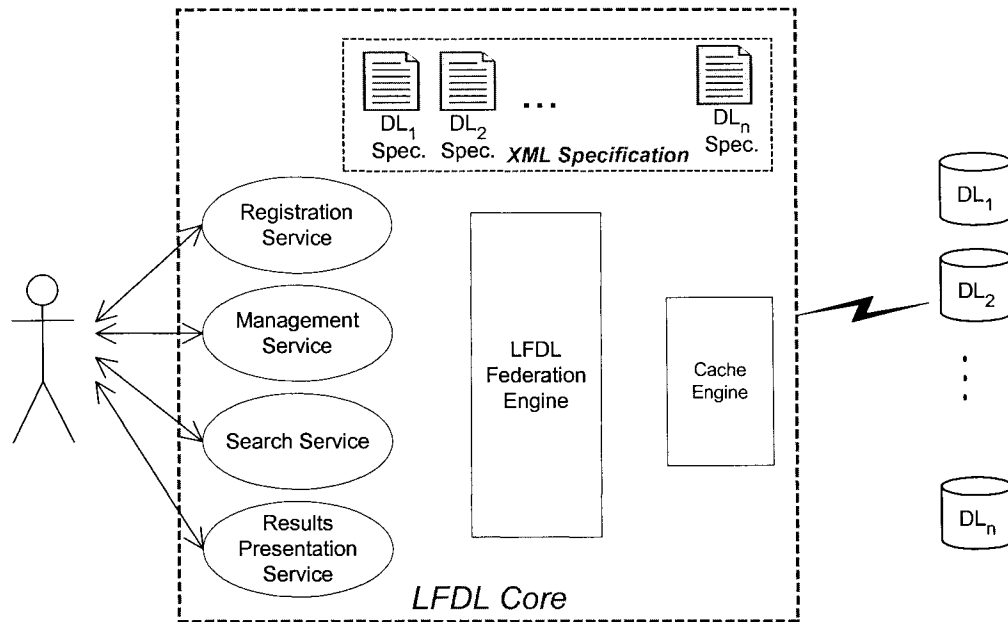


Fig. 3.2. LFDL architecture.

3.2.1 DL Specification

The key to a lightweight, data-centered and rules-driven approach is the interoperability information described in the specification of a digital library. Within a group of heterogeneous digital libraries, each one of them is unique in terms of its search interface and results presentation. Developing code for each DL to wrap up the difference is an option to achieve interoperability. However, that is not flexible and not efficient.

Instead of writing specific code for each DL, we provide a standard specification format or common rules to describe each DL's characteristics, capabilities, and interoperability information using XML. We observe the user interaction with the DL and specify all possible user/DL interactions including query submission and the way a DL presents the results of a query to the user. A DL specifies its unique information, including how its query string format is mapped to the LFDL query format, and special instruction to process its search results, following the common LFDL schema so that the generic LFDL code can enforce the rules.

3.2.2 Registration Service

As a federated service provider, the LFDL needs to be aware of the existence of a DL repository. The registration service allows a new DL to be added to the LFDL federation by registering its specification. The specification can be stored in a centralized server. The format of the specification must follow the standard schema, and the LFDL will check its validity before a DL can be successfully registered.

Either an individual DL expert or an LFDL expert can access the registration interface to register that DL. Once registered, a DL's specification is parsed and stored so that the LFDL Federation Engine can enforce the rules specified in the specification.

3.2.3 Search Service and Results Presentation Service

To end users the LFDL is an enhanced mega search engine. Using a universal or unified search interface, users can send search requests simultaneously to all digital libraries in the federation. The search results will be returned to users as if they are from the same source. The LFDL Federation Engine showed in Figure 3.2 utilizes the specifications of federated DLs to provide the search service and results presentation service.

The details about the data flow and interaction among various LFDL modules to service a search request are as follows:

1. At initialization the system reads all specifications of registered DLs and creates the query mapping rules and results handling rules.
2. A resource discovery user submits a query using the LFDL unified search interface.
3. The query is passed to the LFDL Federation Engine.
4. The Federation Engine uses the query mapping rules to transform the universal query to each DL's native local query.
5. The transformed query is sent to each remote DL and the search results are gathered.
6. The Federation Engine parses the search results pages, using the results handling rules of each DL, and extracts the results.

7. Parsed results from all DLs are merged and displayed to end users.

3.2.4 Management Service

A well-managed information system can achieve desired functionality and improved performance and efficiency. Without proper tool support the management task can be time-consuming and error prone [52]. For the LFDL we design and implement a monitoring and management service to facilitate the needed tasks. A Web interface enables the LFDL managers to start/stop the service and track system runtime information such as each DL's availability and various system statistics, including average system response time, resource usage, and user search behavior data. The management service also allows for fine-tuning the system by adjusting runtime parameters, for example, allocating more system memory for caching.

3.2.5 Caching

System performance and reliability are major problems with DL interoperability approaches using results gathering and distributed search. As the search request is sent simultaneously to multiple DLs and each of them has a different response time, the federated DL's response time is not guaranteed; usually only when the last DL returns something can the end users see the results.

In our approach we provide universal access to heterogeneous DLs at a relatively high level, i.e., a common data model to map high level query language. The communication protocol to each DL is HTTP [30], [31]. Its efficiency depends upon the network traffic, as well as the response time of each remote DL system (determined by its service implementation, the power of the server, and server load). Ideally, such interoperability should be at all levels like in a fully cooperative federation, e.g., using common data query languages, data manipulating and accessing mechanism, the data model, communication protocols, and more. Such integration is currently impossible due to the many autonomous DLs. One of the biggest issues of this high level integration is the sacrifice of performance: without the full control of the query structure, there is no guarantee of the query response time. Therefore, in using a universal interface to access

different DLs, there will be different response times. Additionally, the response time is unpredictable, as we have no full control. When a search is against several individual DLs, the response time the user feels is always equals to the response time of the slowest DL. For an online application like digital library, where response time is critical, such a slow system is unacceptable.

An effective way to improve service performance is by caching search results. Pitkow [97] presents a caching algorithm targeted at WWW-based information system. We can also take advantage of caching to improve the performance of the LFDL system. In addition, a well-designed cache makes the LFDL more robust, flexible and scalable. We propose a LFDL cache (discussed in detail in Section 7) which holds recent search results in local storage. When a search request is served, the cache will be checked first to see if the query and its results have already been kept locally. If there is cache hit, the results will be returned to users instantly. With caching we can also implement asynchronous search and progressively results presentation: instead of waiting for all results come back from all DLs, partial results can be displayed to users first and whenever there are new results available the results-displaying page will be refreshed accordingly.

The LFDL Cache Engine (Figure 3.2) is responsible for the cache to be working properly. For example it enforces the cache size as planned and if the cache is full, it will replace existing entries with new ones according to a cache replacement algorithm (discussed in Section 7). It also maintains a list of cache-miss queries so that a DL Agent can refer to the list and access a remote DL to fetch results for the query that has no results in cache.

Figure 3.3 demonstrates a cache-based search scenario: a user wants to search for the keyword “XML” against the IEEE and the ACM digital library.

- 1) User sends the search request to LFDL web server.
- 2) The LFDL Federation Engine checks against cache.
- 3) For IEEE, the cache finds an entry with the same query string; for ACM, the cache misses.
- 4) Web server returns the result page: for IEEE the search results, for ACM the message “still fetching data from remote DL”.

- 5) An entry is added to a cache-miss list: “XML” for ACM.
- 6) The Federation Engine reads in each entry of the cache-miss list.
- 7) The engine will send the request for “XML” to the ACM digital library and then receive the search result.
- 8) The cache will be updated with the new result. The entry will be deleted from cache-miss list.

In the meantime the web browser will continue sending the request for “XML” against ACM automatically, until the cache hits or times out.

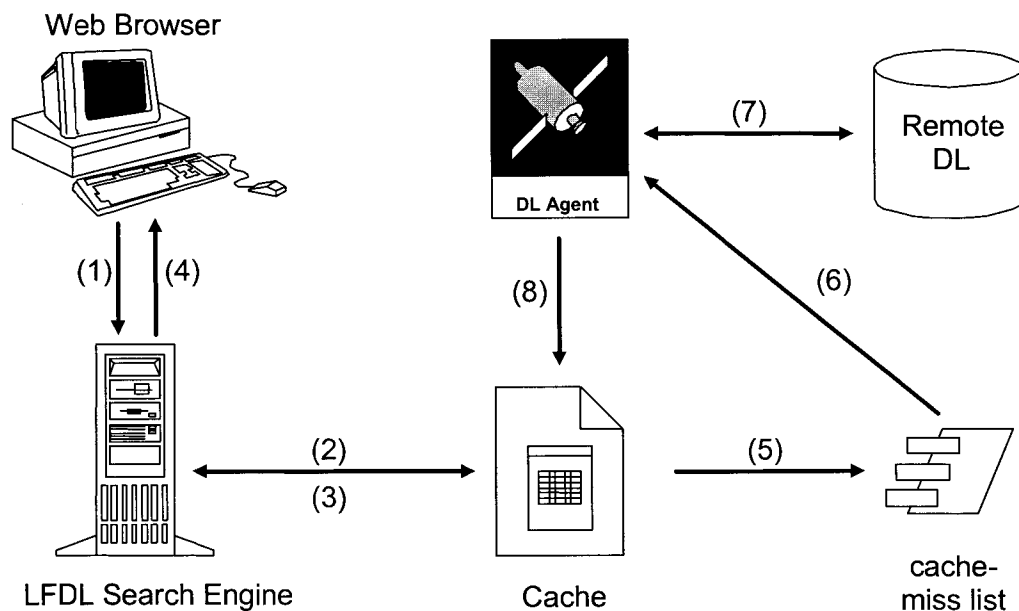


Fig. 3.3. Caching usage scenarios.

3.3 LFDL IMPLEMENTATION: RAPID PROTOTYPE SYSTEM

We realized the first implementation of the LFDL as a rapid prototype system [109], [130]. Three specification documents specifying the three initial libraries (ACM, IEEE, and NCSTRL) are registered. This prototype showed that the LFDL provides a feasible approach in achieving interoperability among non-cooperating digital libraries at least in principle.

The screenshot in Figure 3.4 shows the universal search interface of the LFDL prototype.

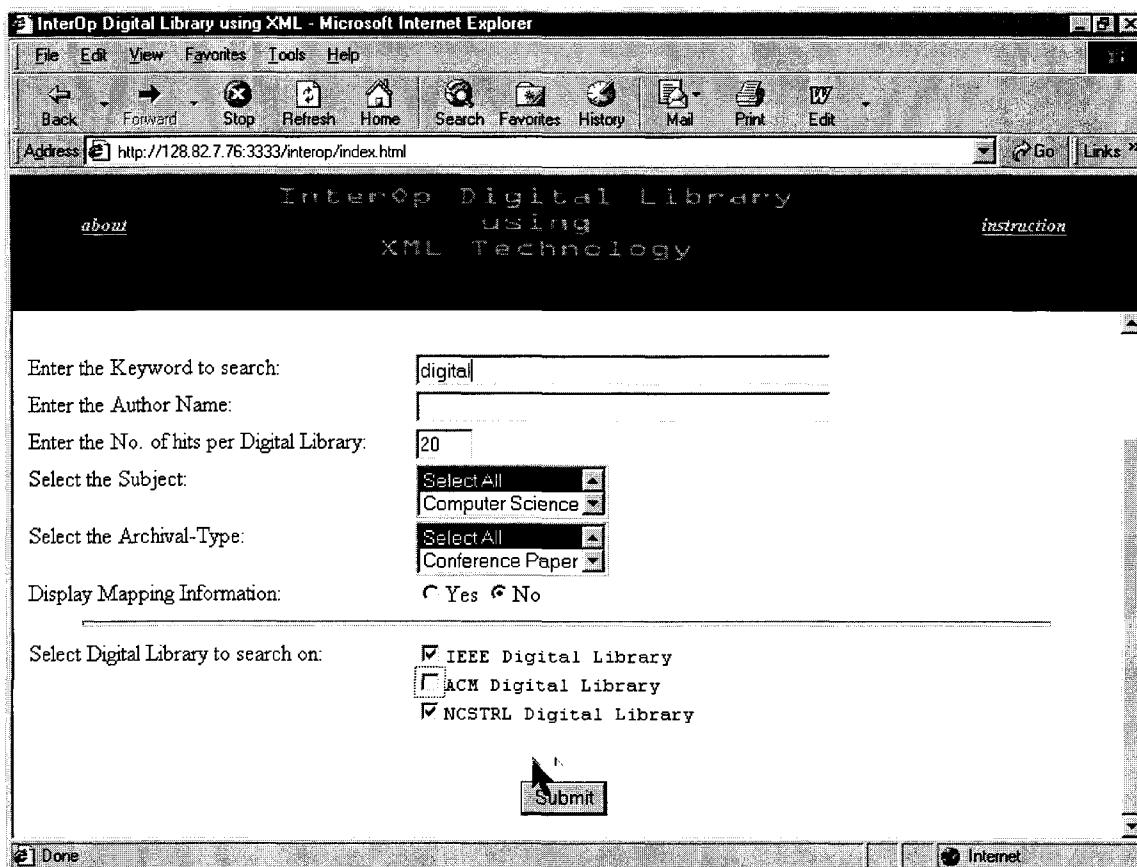


Fig. 3.4. Universal search interface of the LFDL rapid prototype system.

The core LFDL federation engine is implemented as a Java Servlet [38], [47] running on an MS IIS web server. There are many advantages to using Servlets rather than other web application technologies such as CGI. Among them the most important ones are:

- Advantages of Java language: such as platform independence, write once run anywhere, and multithreading [5].
- More efficient: unlike CGI [16], which starts a new process upon receiving a new request, a Java servlet starts only once during its whole life cycle. Whenever there is new request, a thread will be generated to handle the request. This is more

efficient and can achieve better performance. Although FastCGI [15], as an improvement to CGI, addresses the process proliferation issue, it still lacks the efficiency a true multithreading solution that servlets can provide. Both CGI and FastCGI are not trivial to program. Servlets can also take advantage of the benefits of the more open, portable Java language.

- Others: like build-in session control, authentication and security support of the Servlets engine.

All these benefits make Java Servlets an ideal middle tier solution for advanced web-based application system.

The web server provides the common middle tier or interface between the client and the backend services. The requests for registering a DL, or the query for a DL, are all sent to the web server from the client's browser machine. The web server forwards the request to the appropriate service provider, and then sends the results back to the browser.

3.4 DISCUSSION

The LFDL rapid prototype system implementation and test bed demonstrate that the LFDL provides a feasible, lightweight approach to achieve interoperability among non-cooperating DLs. At this stage we were not concerned with efficiency in terms of user response time but more with seamlessness and an engine that is driven by specifications and not by specific codes for different DLs.

In our prototype implementation we have the feature to allow for dynamic additions of new libraries. In various experiments we have been successful in showing the power of our approach. We started with including only IEEE in our LFDL and once a user submits a query and LFDL returns the appropriate results as from the IEEE. Next we defined a specification for the ACM DL and added the description to the LFDL using the registration service. After reissued the query without any code change and the LFDL produced the query results from both ACM and IEEE. It should be clear that the LFDL prototype did not post process search results and only presented results in their native format as returned by the participating DLs, usually a list of document records and each has a clickable hyperlink. Once a user picks a particular record by clicking on its link, he

will be redirected to that digital library. The LFDL prototype by itself does not maintain any record locally and only serves as a broker [132].

The prototype system has limitations in terms of search capabilities, service usability, quality of service (precision/recall), and performance. In the following chapters we will give details on how we design and implement the various LFDL services to address those limitations and evolve the prototype LFDL into a useful system.

SECTION 4

DATA-CENTERED RULES-DRIVEN INTEROPERABILITY: DL SPECIFICATION

In the previous section we introduced the basic approach and the overall architecture of the LFDL in building an interoperable federation of heterogeneous digital libraries. The essential part of the LFDL is DL specification, which describes a DL's interoperability information.

In this section we introduce a XML-based Digital Library Definition Language, or DLDL, to describe the methods to interact with a digital library. The section is organized as follows:

- In section 4.1 we present an overview of the data-centered interoperability of the LFDL.
- We then in section 4.2 discuss the design and implementation of DL specification schema based on the DLDL.
- Section 4.3 describes how to use the DLDL to compose a DL specification.
- Finally section 4.4 discusses issues of using the DLDL for a highly-diverse collection of digital libraries.

4.1 INTRODUCTION

Our interoperability approach is based on a data-centered rules-driven architecture that allows individual DL system to describe itself so that a federated service can be built by enforcing the rules specified in the description. The federation supports a unified interface that allows users to search participating digital libraries and get results that are dynamically constructed depending on the profile of the target DLs.

In this approach the inside architecture and implementation of each DL is invisible. Not only it is convenient for an existing independent DL to join the federation, it also alleviates the management and maintenance burden of the federated service provider. In stead of writing code for each DL for its unique features, we develop the generic LFDL

Federation Engine and use self-described rules specified in a DL's specification to build the integration. Any changes in a registered DL can be handles easily by updating the DL specification - no code change, no redeployment.

The key is how and what to define in the specification to describe all interoperability related information of a digital library. Though different DLs have varying degrees of "openness", they all have to provide at least a search interface and results display interface for end users to utilize the services. Some do provide browsing services for users to scan through collections but in this dissertation we focus on the search related services. Thanks to the popularity of the Internet and WWW, the majority of the interfaces are Web-based instead of being implemented on proprietary systems. DLs' native search and results presentation interfaces are the sources for the LFDL to build a federated search service. A DL's profile specification is thus served as the mapping between the LFDL unified search interface and results display interface. As illustrated in Figure 4.1 a DL's specification includes the query translation rules which specifies the mapping between an LFDL query and DL native query, as well as results processing rules on how to trim unrelated information from the DL's results page and fetch the actual search results.

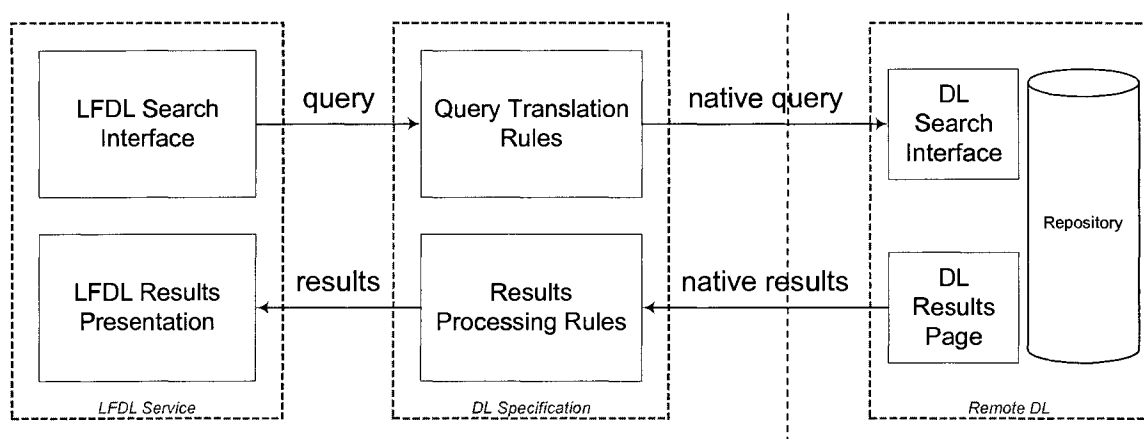


Fig. 4.1. Specification based LFDL federation.

4.2 DIGITAL LIBRARY DEFINITION LANGUAGE (DLDL)

To create a specification for a DL we need a schema to define a common set of rules and standard format. That is, different DLs have to “speak the same language” when describing their specific features. This enables a single generic LFDL Federation Engine to read in a DL specification and enforce its rules.

To design and implement the schema one option is to use a traditional relational database and define a set of table structures. However, a database-based schema is not flexible and not easy to maintain or update. To read, enter or update data, code has to be developed to provide a human-machine interface so that users can access the data in database. In case to modify the schema, the whole database table structure may have to be changed, as well as the data manipulating code.

Ideally, the schema, as well as the specification, should be simple, straightforward, human readable, and easy to modify. The Extensible Markup Language (XML) [125], [40], which is a simple dialect of SGML [116] and has been endorsed by W3C [124], provides an ideal solution.

XML transforms data in a format that can be easily processed between different organizations each of which has its own data format and structure. XML makes data portable and independent of implementation by making data self-describing. XML provides a non-proprietary way to label data objects and it provides a universal syntax for representing the structure and description of data, indifferent to application logic. XML allows exchange, sharing and use of data across applications, organizations, and platforms in a standard, cost-effective way over the network. This exchange lets developers write applications that can run on any platform and let everyone view and leverage data similarly, regardless of system or operating environment. XML is a flexible language that can easily accommodate changes. There are many parser tools available and it can be used on multiple platforms. In addition an XML document is human-readable and can be edited using any text edit tool.

XML Data Type Definition (DTD) [101] or Schema [119], [126] is a perfect match for DL specification schema. A DTD defines the structure of an XML document and it allows the XML parser to check whether the document is valid or not and whether it is

well formed. XML Schema is an alternative to DTD, which unlike DTD, is itself XML based. XML Schema is more extensible and richer than DTD.

We develop the XML-based Digital Library Definition Language (DLDDL) to specify the externally observable behavior of a DL; that is, for each DL an XML description is used to define the metadata of that DL, or define the form that the DL expects queries in and how it presents the results to the user. In DLDDL we use a DTD to define DL specification schema because it is simpler than a XML Schema and enough for this application. Figure 4.2 lists part of the DTD which lists three important piece of information of a DL specification: the content of this digital library, methods to access the digital library, and what information must be retrieved from the digital library.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE DLDDL[
<!ELEMENT DLDDL (TITLE, DOCID, BASEURL, DLIBINFO, SEARCHDATA)>
...
<!ELEMENT DLIBINFO (ORGANISATION, ARCHIVAL-TYPE*, SUBJECT*)>
...
<!ELEMENT SEARCHDATA (REPLACE-FIELD, SEARCH-METHOD, SEARCH-
URL, INPUTDATA*, OUTPUTDATA*, MULTIPAGE)>
...
<!ELEMENT INPUTDATA (IVARIABLE-NAME, IVARIABLE-TYPE, FORMNAME,
DEFAULTVAL, REPLACE-NUM)>
...
<!ELEMENT OUTPUTDATA (OVAR-TAG, OVAR-MATCH)>
...
<!ELEMENT MULTIPAGE (MULTI-PAGE, HAS-NEXT, NEXT-URL, LINK-URL,
PAGE-HIT)>
]>
```

Fig. 4.2. Part of the DTD of a DL specification.

4.3 DL SPECIFICATION DEFINITION USING DLDDL

Now that we have the specification schema defined using DLDDL DTD we can start to use it to describe a digital library's metadata information. In the following sub sections we describe each DLDDL XML tag and its usage. More details can be found in [130].

4.3.1 Digital Library Content

This set of tags gives general information or metadata about a DL such as its title and URL. The TITLE attribute of a tag describes the function of the tag. The tag DLIBINFO has three additional tags:

<ORGANISATION>: The organization that maintains this digital library.

<ARCHIVAL-TYPE> and <SUBJECT>: The type of materials the DL consists as well as the DL's general subject category. These tags are for information only currently. In the future they can be used for the field mapping.

4.3.2 Digital Library Access Methods

This set of tags specifies the rules on how to access a remote DL as well as how to map a LFDL unified query to that DL's native query. The tags can be divided into different sections. The first set of tags give information on the location and search method of a digital library's search service. An example is shown in Figure 4.3:

```
<SEARCHDATA Title="Search Info:">
<SEARCH-METHOD Title="Search Method:">POST</SEARCH-METHOD>
<SEARCH-URL Title="SearchURL:">http://www.acm.org/ows-
bin/dl/owa/dl.search</SEARCH-URL>
```

Fig. 4.3. Specification sample: remote DL access information.

The SEARCH-URL tag indicates the URL of the search interface of a digital library server and the SEARCH-METHOD tells the access method to the HTML form of the search interface. The standard POST or GET method can be used. The above sample shows that the ACM digital library uses POST method to submit a query to its search service.

For the LFDL to access a DL, it has to know the search interface of that DL. And the format of the LFDL query string has to be translated to the native format of that DL so that the LFDL can distribute its query to the DL. Figure 4.4 demonstrates the set of tags that describe a DL's search interface information and how it can be mapped to the LFDL

universal search interface. For example, the HTML form of the ACM digital library has a text input field which is displayed as “Search DL”. Its interior query string name is “query”, which can be mapped to the LFDL query string name “UI_keyword”. Therefore, when a user search for “computer” using the LFDL universal search interface, the LFDL can look at the specification of ACM and translate LFDL query string “UI_keyword=computer” to ACM native query string “query=computer”.

```

<FORMFIELD>
  <INPUTNAME>
    <LABEL Title="Displayed Field Name:">Search DL</LABEL>
    <INPUTNAME_VALUE Title="Internal Form
Name: ">query</INPUTNAME_VALUE>
    <INPUTNAME_MAPPING Title="Mapped UI Field
Name: ">UI_keyword</INPUTNAME_MAPPING>
  </INPUTNAME>
  <INPUTTYPE Title="Form Type:">text input</INPUTTYPE>
  <INPUTVALUE/>
</FORMFIELD>

```

Fig. 4.4. Specification sample: DL search interface information.

4.3.3 Information to be retrieved from Digital Library

This set of tags gives information on how to parse the results from a digital library. The DL output is an HTML page that contains the required links to the desired documents. However that HTML page is often not well constructed and it contains many unrelated links. We need the following information from that HTML page: the correct links to the valid search result documents and whether the results are returned on one page or multiple pages. If it is returned on multiple pages then we need the necessary information to retrieve all the result pages. The set of tags in Figure 4.5 tells how to parse and get the correct links to the documents. The OVAR-MATCH tag indicates the matching string for a result document. The OVAR-TAG tag specifies the HTML tag to be searched to see if it contains the matching string.

```

<OUTPUTDATA Title="ACM Output:">
<OVAR-TAG Title="Output Tag:">A</OVAR-TAG>
<OVAR-MATCH Title="Output Match:">/pubs/citations/</OVAR-MATCH>
</OUTPUTDATA>

```

Fig. 4.5. Specification Sample: results matching information.

The set of tags in Figure 4.6 specifies how to parse DL search results that may be stretched across multiple pages. If DL output is listed on a single result page the value of the MULTI-PAGE tag is "no" and the rest of the tags have "null" as their value. If DL results are displayed on multiple pages all of the HTML pages have to be retrieved. A "yes" value of the MULTI-PAGE tag indicates multiple results pages. The HAS-NEXT and NEXT-URL tags are for the case that there is a link to the next result page and the following page has a link to the next page and so forth. The more common way is to have links to all the remaining pages on the first page and the tag LINK-URL gives the matching string of the links to the other pages. The PAGE-HIT tag tells the number of hits that is returned on one single page so that the number of pages to be retrieved can be limited based on the number of hits a user wants.

```

<MULTIPAGE Title="Multi Page Information">
<MULTI-PAGE Title="MultiPage:"> yes </MULTI-PAGE>
<HAS-NEXT Title="Contains Next Link:"> no </HAS-NEXT>
<NEXT-URL Title="Matching String:">null</NEXT-URL>
<LINK-URL Title="Matching String:">/ows-
bin/dl/owa/dl.result_page?search_conid </LINK-URL>
<PAGE-HIT Title="No. of hits per page:">24</PAGE-HIT>
</MULTIPAGE>

```

Fig. 4.6. Specification sample: multiple results page information.

4.4 DISCUSSION

We have defined and registered the metadata specification for half a dozen DLs using the DLDL. The registered DLs are all quite different, not only in content but in organization and implementation as well. These DLs have different search interface, reliability, and response time. All these point out that there is considerable peril in attempting to federate heterogeneous libraries. The LFDL test bed demonstrates that the DLDL is capable of grasping the essential DL interoperability information, and it is flexible in the sense that it allows a large variety of digital libraries to have an XML specification which can be used with the LFDL search software. The self-described XML specification based on the DLDL is simple to read and a user can easily edit it. Also, the LFDL's data-centered architecture fits well in the more popular distributed, inter-organizational, web-based computing model such as the Web Services [122].

A similar approach is described in [99], in which a Searchable Database Markup Language, SearchDB-ML based on XML is defined. This approach differs in that it is targeted for Web sites that support simple search interfaces rather than libraries with support for clustering and advanced searches, and it does not support dynamic discovery and integration of a digital library in the federation. Lyceum [72] is another earlier data driven approach in which a query gateway or meta-search engine provides a unified interface to heterogeneous and distributed information resources, though it is pre-XML and of relatively smaller scope than DLDL. Target mainly Web sources, DEByE or Data Extraction by Example [24], is a tool for extracting hidden Web data based on user specified examples.

One issue we have to point out is the intellectual property right, which prevents many DLs like ACM and IEEE from cooperating in the first place. For the LFDL although we talked to publication officials of the ACM library we did not do so for other DLs. The assumption that what is available on the Web for free can also be included in our federation may be wrong. We feel that in many ways we use the same rights the general Web indexers use. At this stage we are not addressing intellectual property directly, considering our work currently is still mainly for research purpose. We do think it is an important issue to be discussed in the future. Also, once the federation service is finalized and deployed successfully in production, organizations may be willing to participate.

The diversity of digital libraries makes it unrealistic to design a ubiquitous schema that can describe all types of digital libraries or the very single aspect of a DL. In the following section we discuss various other issues of the design and usage of the DLDL.

4.4.1 DL Search Interface Capture and Query Mapping

One of the basic functions of the DLDL is to describe the search interface of a DL and define rules of query mapping between the LFDL and native DL. It is possible that some features of the search interface presentation within a given DL are not captured by the DLDL. For example some HTML form fields of a DL may have pre-defined option values, but the DLDL does not define any common values and therefore there are no mappings to DL specified values.

The DLDL is capable of specifying the differences in syntax of a DL's search fields and filters; however, the current schema does not resolve the different semantics of the search interfaces of different DLs. The syntax differences may be simply that the number of fields is different or that the naming of the corresponding fields is different. For example one DL may not have a "title" search field and another DL may have named the author field as "creator" field instead. Using the DLDL we can create the specification for a DL so that each of its fields is mapped to the generic LFDL search field.

The subject clustering mapping problem [133] represents the general semantics mapping issues of the DLDL. Semantic differences may also occur when one DL returns, for example, an undifferentiated, unchecked character string for a date field, whereas in the universal LFDL we consider date to be an object that can be read by a standard calendar program.

The semantic differences in mapping are not easy to solve, considering various DLs may have numbers of fields that have different pre-defined values. One option is, in addition to the specification per DL, to create a generic meta-tag specification for the LFDL [133]. The generic LFDL specification will define all possible DL filter values in a universal, neutral format. Each DL's specification can then map each of its filter values to the generic one. We need to do more research to find out if this universal, maximal specification is workable and economical.

Another major problem is how to describe a non-web form based interface, in which other methods like a Java applet instead of a form are used to present interfaces for user input. Similar problems occur when we want to incorporate multimedia digital libraries: what if a clickable map is used to send a query? Though those are not common scenarios in a digital library, we have to be aware of them. Until now we have not addressed them in our approach, but we may explore those issues in future work.

4.4.2 Search Process Simulation and Specification

There are some other issues related to the simulation of different processes or patterns that a particular digital library supports and users have to follow to place a search, e.g., access control and multi-step search. The difference in access control mechanism is a major issue that we have to deal within the LFDL. The problem arises when some or all DLs only allow access upon some sort of user authentication. The question is, how to integrate the particular process in the LFDL in such a way that the user does not have to deal with multiple different authentication processes but only one (possibly involving several passwords). A similar problem is posed when a particular DL partitions a search into separate stages going back between the server and client to achieve a particular query. Consider a DL that allows a user to specify a subject taxonomy and then makes a selection from the chosen taxonomy.

The following table lists various search process specification issues, sample scenarios, solutions or options. Mostly it requires that we extend the DLDDL schema and then enhance the LFDL Federation Engine to comply with the DLDDL changes.

Access Control IEEE [43] has access control to its search service and the authentication is done by two HTML form fields of userid and password. We can extend DLDDL specification for IEEE to include such information: a “guest” user with password of “welcome”. However, this only works for DLs that have universal user access and it is available to the public for free. We still need to address individual authentication in which each end user has his own id and password.

Session/Cookie Control Many digital library services require that client-side browsers to support HTTP sessions or cookies to finish a search. As a software agent which simulates the browser to interactive with the HTTP server, the LFDL has to take

proper action to access a remote DL which is session/cookie based. We have modified the DLDL and the LFDL Federation Engine implementation to support this.

Redirected Service The LTRS [59] digital library redirects a user query to another address to fulfill the search request. We can easily make code change to the LFDL engine as the underlying Java network package used by the LFDL supports the option to follow redirected HTTP links.

Linked Link Page If search results of a user query spread out multiple pages, usually a DL displays all links to the other pages on the same page which displays the first set of search results. But the LTRS presents just one link to the next page on its current result page, so users have to browse results sequentially but cannot jump to a result page randomly. We also added this information to the DLD so that the LFDL can act accordingly.

TABLE II
PROCESS SPECIFICATION: OTHER ISSUES

Problem	Sample	Solution or Option
Access control	IEEE DL password protected	<ADDITIONAL name="userid">guest</ADDITIONAL> <ADDITIONAL name="password">welcome</ADDITIONAL>
Session or cookie control	ACM, Arc, NEEDS	<ADDITIONAL name="usecookie">true</ADDITIONAL>
Redirected service	LTRS redirects query to inktomi search engine	Code: follow redirected link
Linked next page links	LTRS	<ADDITIONAL name="linked link page">true</ADDITIONAL>
multiple display for single metadata field	Arc	<ADDITIONAL name="multiple record page metadata matching">true</ADDITIONAL>

The last issue we want to describe pertains to query optimization and DL capability description. When presenting the user with a choice as to which of the participating DLs to include in a search, it will be useful to somehow describe in a concise way the capabilities of that DL and also its content and management policies. That same information can also be used when a user makes a search on all DLs. For a good response time it will be essential for the LFDL to use this information in setting filters and selecting DLs to be part of the search. It does not make sense, for example, to search arXiv.org - a physics collection - when the subject selected is arts.

SECTION 5

SEARCH SERVICE: USER-CENTERED DYNAMIC SEARCH

One major objective of digital library interoperability is to provide a federated search service so users can utilize a unified interface to search multiple collections at one time [67]. This section introduces the effort of building the LFDL federated search service. To improve service quality and usability we present a user-centered, need-driven, interactive search mechanism based on a dynamically generated user search interface.

The remainder of this section is organized as follows:

- Section 5.1 introduces the challenges of building a unified search interface across heterogeneous digital libraries.
- In section 5.2, we discuss an advanced, interactive search approach to build the LFDL federated search interface. The interface is generated dynamically based on user's search need and the profiles of the digital libraries that are related to the query submitted by the user.
- Section 5.3 analyzes experiments with our implementation.

5.1 INTRODUCTION

In the LFDL the Federation Engine implements the end-user search service and results presentation service. Figure 5.1 lists the major components of the engine and the data flow among them to fulfill a user query.

As illustrated in Figure 5.1 at the back end each participating DL registers its specification by giving its metadata description and access rules to the LFDL rules engine. The LFDL search engine coordinates with the rules engine to provide the search service. A DL Agent is the mediator between the LFDL and a remote DL. It is created based on a DL's specification once it registers with the LFDL and has the information on how to communicate with that DL: where and how to send a translated query to the DL, as well as how to interpret the results back from that DL.

On the front end end-users employ the universal search interface to access the federated search. Once a user submits a query, the search engine will use the query

mapping rules from the rules engine to translate the query to a DL's native query. The DL agent will then send the translated query to a remote DL and get results back. The result process engine will parse the result set and save it to cache and then display it to end-users.

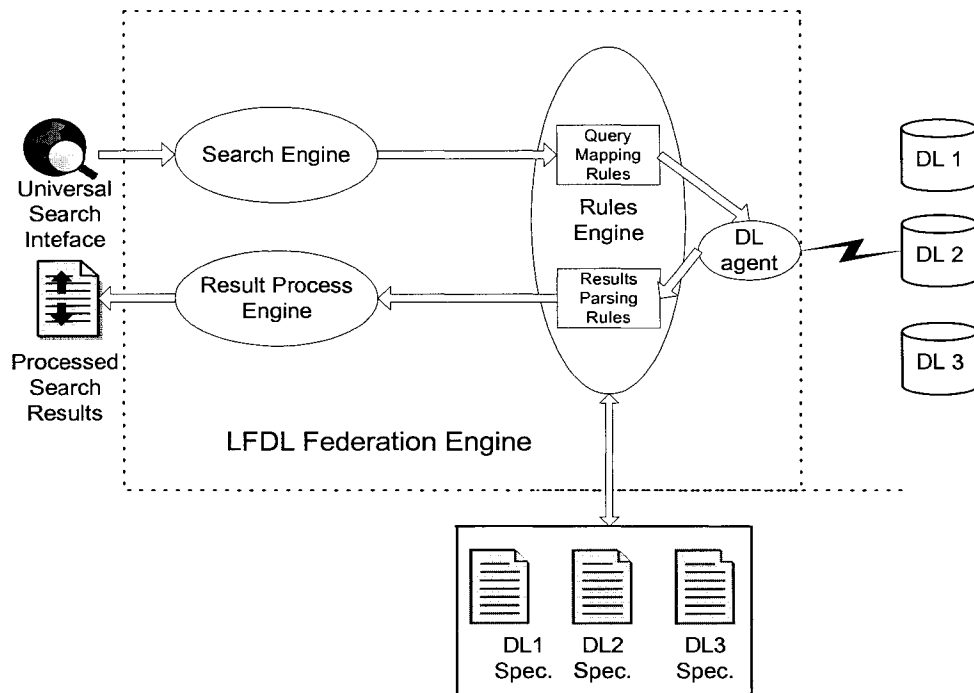


Fig. 5.1. Federated search service and data flow.

The details about the data flow and interaction among various LFDL modules to serve a search request are as follows:

- 1) At initialization the system reads all specifications of registered DLs and provides the rules engine with query mapping rules and results handling rules. Also for each DL a DL Agent is generated.
- 2) A resource discovery user submits a query using the unified search interface.
- 3) The query is passed to the search engine.
- 4) The search engine works with the rules engine and uses the query mapping rules to transform the universal query to each DL's native local query.

- 5) Each DL Agent sends the transformed query to the remote DL and receives the search results.
- 6) The result process engine parses the search results pages, using the rules from the rules engine, and extracts the results.
- 7) Parsed results from all DLs are merged and displayed to end users.

A major challenge of any federated service is to present a single, unified user interface that maps a user's selections for various fields in the search options to queries for the participating libraries [133]. It is a difficult task to design user-friendly, advanced search interface for a digital library so that users are willing to use it to search the resource for more accurate results. In the LFDL rapid prototype system implementation (see Section 3), we had established a simple-static interface. To enhance the LFDL we shall address the problems mentioned above. That is, design an advanced search capability and induce the user to take advantage of it. We propose a dynamic LFDL interface, one that is customized based on the user's selection of libraries and the type of material the user is looking for [110].

5.2 APPROACH, DESIGN, AND IMPLEMENTATION

Traditional advanced search interfaces assume users are able and willing to constrain their search by entering/selecting values in various fields and filters. We believe this is not a valid assumption, because for most users this is too time consuming and the design is often too confusing, requiring explanations to be checked before values are entered. Most users prefer to use a simple keyword based search interface [110]. In our approach we have provided a series of interfaces, starting with a simple keyword search interface. Based on the users input for the first interface, we tailor and fine tune the next interface so that only the essential filters of libraries with content related to the keyword will be presented and all irrelevant features will be omitted. For the sophisticated user, we allow for the customization of the interface on the fly so that other features can be selected in addition to the ones selected automatically by the system.

5.2.1 User-centered, Need-driven Search Mechanism

Our solution to solve the unified interface presentation problem is based on the user-centric approach where users engage in a series of interactions with the federation service to finish a search. There are two phases of interactions to submit user queries. In the first stage a user submits a keyword, and in the second stage, a dynamic generated interface with filters related to the query will be presented. The user can then utilize the filters desired to submit the query.

The basic idea is to maintain a large keyword set and associate a relevance or weight to each DL with each keyword. If a keyword is more relevant to DL A than DL B, the dynamically generated search interface should reflect more features from A than from B. This way a more accurate search can be sent to the DLs more related to that keyword and a higher quality of service can be provided to users.

The keyword set can be created from two sources: analyzing all metadata in the archives of the federation and analyzing the logs of users of the federation. A problem exists in initializing this set before the federation is in a steady state and another in obtaining all the metadata of a participating DL that is generally not available – remember, we do not rely on member cooperation. Here we are presented with two issues: the need for a base keyword set, and the need to calculate the relevance for each DL for each keyword in the set so that we know which DL has matching records.

To generate a base keyword set, we utilize Arc, a federation of over 100 digital libraries [69]. These DLs provide all their metadata to the federation following the OAI-PMH and at ODU we maintain a repository database to store the metadata. We have designed and implemented a process that goes through the Arc metadata repository and then calculates the most frequently occurring (in the metadata records) keywords. The results will be stored in a relational database. Considering the DLs registered with OAI are across quite different disciplines, the keyword set generated from those DLs is reasonably representative. This federation also keeps users' logs that we analyzed. The following table shows the top 10 keywords as well as the number of occurrences from selected DLs.

TABLE III
KEYWORDS AND NUMBER OF OCCURRENCES FROM THE DL
METADATA DATABASE

Arc metadata		Cogprints metadata		LTRS metadata		WCR metadata	
STATE	14977	PSYCHOLOGY	1968	ANALYSIS	212	WEB	194
STATES	12038	PHILOSOPHY	998	SYSTEM	206	CHARACTERIZATION	137
UNITED	11479	NEUROSCIENCE	824	MODEL	193	WORLD	27
HISTORY	11032	SCIENCE	747	AN	187	WIDE	25
EDUCATION	7459	COMPUTER	613	DESIGN	169	CACHING	17
PSYCHOLOGY	4956	COGNITIVE	542	CONTROL	157	TRAFFIC	16
CRITICISM	4578	BIOLOGY	406	FLOW	146	SERVER	14
STUDY	4192	LINGUISTIC	279	USING	138	PROXY	13
TEACHING	3937	LINGUISTICS	269	HIGH	127	WWW	12
LANGUAGE	3763	MIND	263	FLIGHT	124	CACHE	11

Once this base keyword set is defined, we determine the relevance or weight of each keyword within the set by sending each keyword to each participating DL. We associate with each keyword the number of hits a DL produces for that keyword. Thus, each keyword has for each DL a weight associated. A more accurate weighing algorithm could be Term Frequency Inverse Document Frequency (TFIDF) [103] based strategy. However, most DLs only expose keyword hits information by displaying the number of documents related to a keyword. Therefore, without the knowledge of a DL's internal documents set we can only calculate the weight of a keyword from the hits the DL shows.

We expand the DLDL to include the results parsing rule so that a DL can use it to specify how to extract the keyword hits information from the keyword search result page. The results will be stored in a relational database. As illustrated in Figure 5.2 for each keyword in the base set, a keyword-relevance fetching agent sends a request to a DL. Based on the hits parsing rule of that DL, the agent collects the result page and extracts document hits for that keyword and saves the mapping keyword-hit number into the database.

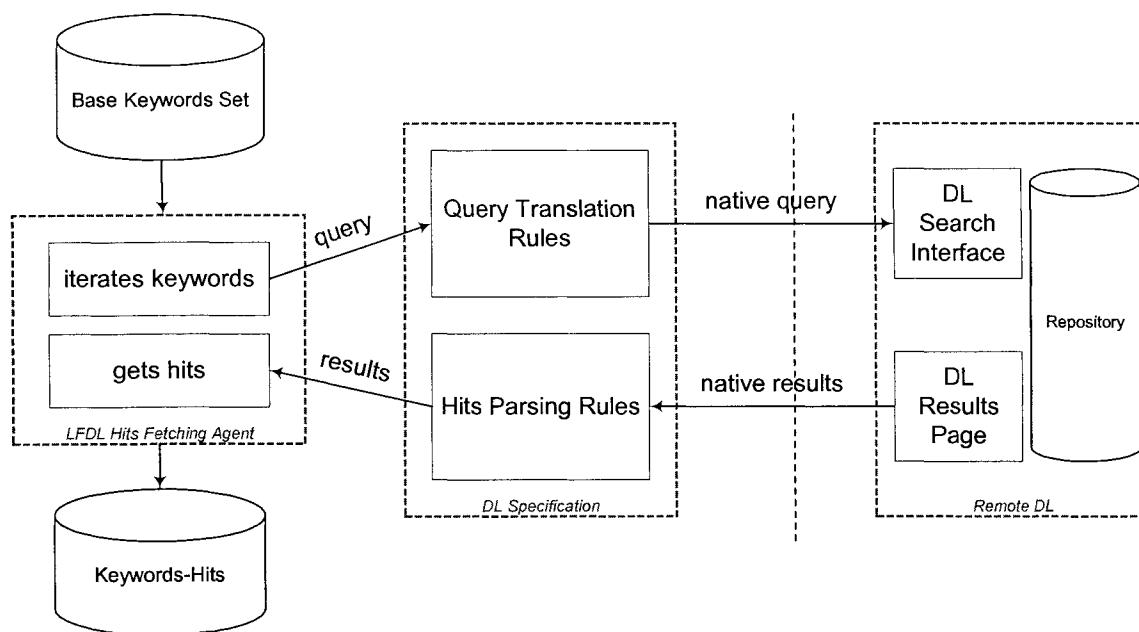


Fig. 5.2. Populating keywords-hits for a DL.

Since this is a time consuming process (and potentially taxing to the participating DLs) we would not do this as a real-time process but more likely on a daily or even weekly basis. The documents hits information for a keyword is fairly static for a library and the keyword list itself should not change dramatically once a steady state has been reached. Some DLs may restrict or refuse to service this sort of heavy load placed by an automatic agent or robot. We can adjust the agent visit interval and/or if possible coordinate with target DLs so that they allow the access.

Table IV shows some top keywords with the highest hits for selected digital libraries [18], [43], [84], [128].

Based on keywords-hits information, a dynamic, interactive interface can be presented. First we use an algorithm to decide which DLs have the most relevance and then we select which advanced search interface features of the most relevant DLs to include in the universal user interface. To make this an effective procedure we need a) a generic universal interface or UI, and b) a complete specification of all search features of the participating DLs.

TABLE IV
TOP KEYWORD-HITS FROM SELECTED DLS

COGPRINTS		IEEE		NEEDS		WCR	
ART	569	computing	21165	UNIVERSITY	659	DE	90
SYSTEM	384	U.S.	12785	INFORMATION	357	LA	88
THEORY	301	CA	10293	SCIENCE	325	CA	87
RELATION	226	N.Y.	3984	SCIENCE:	325	EL	69
NEW	215	NEW	2707	STUDENTS	293	WORK	52
LANGUAGE	211	SOCIETY	2452	ENGINEERING	292	NETWORK	46
STUDY	192	DE	1887	PHYSICS	270	PRES	40
OBJECT	186	INFORMATION	1690	STATE	266	WORLD	37
SCIENCE	173	LA	1426	COMPUTER	263	USE	35
RELATIONS	156	SYSTEMS	1380	LEARNING	248	ART	34
ANALYSIS	152	HOME	1312	USE	232	RESEARCH	34
STATE	151	ENGINEERING	1279	TEACHING	216	PRESS	32
PSYCHOLOGY	141	TECHNOLOGY	1276	DESIGN	191	AGE	29
CONDITION	136	POWER	1230	MATERIALS	191	RACE	27
SOCIAL	124	TECHNICAL	1171	DATA	188	COMPUTER	27
COMPUTER	106	DATA	1082	NEW	184	RAT	25
ASPECTS	88	TIME	1069	SYSTEM	183	UNIVERSITY	25
STATES	82	COMPUTER	1001	EDUCATION	182	SCIENCE	24
CONDITIONS	72	BOARD	879	TIME	168	VIRGINIA	24
CHILDREN	58	EL	859	PROGRAM	168	PUBLIC	24
PHYSICS	52	CONTROL	805	CENTER	167	GROUP	22

5.2.2 A Generic Base Universal Interface

DL search interfaces vary considerably, and it is almost impossible to create a complete universal interface that includes all features of all DLs. The design goal of the base search interface for a federated service is to create an interface that is as general as possible instead of complete. The Dublin Core or DC [27], [123] metadata set provides an ideal basis to use as filters to create such an interface. Dublin Core defines a common set of metadata. Many digital libraries have either fully adopted DC or provide interfaces using at least several DC elements. Therefore, we chose the majority of the elements in Dublin Core as our basis in defining our UI, with some additional features such as display options or number of hits, which are not in DC context but more important for a federation digital library service.

Figure 5.3 demonstrates the LFDL generic universal search interface. Most of the searchable fields or filters, such as keyword, creator, and title, are directly mapped to the

elements of the DC. The “No. of Hits per DL” and “Criteria combination” are not directly linked to a document’s metadata field, but for users to refine the search.

The screenshot displays the 'InterOp: Universal Search Interface'. It features a series of input fields for search criteria: Keyword, Creator, Title, Description, Publisher, Date, Subject Category, Type, Format, Source, Language, and ID. Below these fields are two dropdown menus: 'No. of Hits per DL' (set to 20) and 'Criteria Combination' (set to Any). A section titled 'Select Digital Library to search on:' contains two columns of checkboxes, each with a digital library name: ACM, IEEE, NEEDS, ANMEN, COGPRINTS, CIAS, CSTC, LTRS, NACA, OTA, OVP, and WCR. All checkboxes are checked. A 'Submit' button is located at the bottom center of the interface.

Fig. 5.3. Generic universal search interface.

5.2.3 Enhanced DLDL and DL Specification

Based on the generic universal interface we enhance the DLDL to have the capability of describing the essential features of a DL’s interface. First, in order to capture those features we conduct a thorough survey of the search interfaces of current digital libraries in the LFDL test-bed. A sample search interface, used by NEEDS, is displayed in Figure 5.4.

Table V lists the search interfaces and features of the digital libraries in the LFDL test-bed; including native DL form fields, their mapping to Dublin Core elements and LFDL universal interface form fields, as well as results output information. For example NEEDS, whose search interface showed in Figure 5.4, has a native search criteria filter

field named Author/Creator. It is a text input type within the HTML form. The internal input form name is author. It can be mapped to the DC element of Creator, and the LFDL UI field of creator.

Fig. 5.4. Native search interface of NEEDS.

TABLE V

SUMMARY OF NATIVE FORM FIELDS INFORMATION OF DLS IN LFDL TEST-BED

DL	UI - common Field Name	Dublin Core element	Field name	Field type	Internal name	Default value (internal value)	Search criteria or display options
NEEDS Results (creator, affiliation, last updated, score)	title	title	title	text input	title		criteria
	creator	creator	Author/creator	text input	contributor		criteria
	publisher	publisher	publisher	text input	publisher		criteria
	keyword	subject	keywords	text input	keywords		criteria

TABLE V (Continued)

CSTC Results(Author, Date, Category, Subject)	keyword	subject	keyword	text input	terms		criteria
LTRS	title	title	Document title	text input	ti		criteria
	creator	creator	authors	text input	au		criteria
	description	description	abstract	text input	abs		criteria
	ID	identifier?	Report Number	text input	rep		criteria
	subject category		Category	single selection	sti	All Categories (*)	criteria
	criteria combination		Select	radio box	boolean	(AND) AND	criteria
NACA	keyword	subject	keyword search	text input	search_words		criteria
OTA	creator	creator	author	text input	author		criteria
			author search type	single selection	authorSEL	(all) all of	criteria
	title	title	title	text input	title		criteria
			title search type	single selection	titleSEL	(all) all of	criteria
	subject category	subject	subject	text input	subject		criteria
			subject search type	single selection	subjectSEL	(all) all of	criteria
	language	language	language	text input	language		criteria
			Language search type	single selection	languageSEL	(all) all of	criteria
WCR Results (type, author, address, date)	keyword	subject	Search	text input	spec		criteria
			Hits/page	single selection	pagelength	20	display options

In general filters listed in Table V are presented as HTML form fields, which typically have type, length, label, and value. To describe all this information as well as

the mapping information to the generic LFDL UI, we expand the DLDL schema so that it can describe HTML form field type (e.g., text input, checkbox, option button, drop-down box selection), field length, displayed field name or label, default values and optional values as well as those values' mapping with the corresponding values of UI fields. We also allow filters that are unique to a DL and have no counterpart in the UI to be specified. These filters will be presented to the user in the generated interface if that DL as well as its filters are highly relevant to the search query. This makes it possible to provide almost the same search quality as accessing each DL directly.

Figure 5.5 lists the DLDL XML DTD for the part of DL search field description.

```

<!ELEMENT INPUTNAME (INPUTNAME_VALUE, INPUTNAME_MAPPING)>
<!ELEMENT INPUTNAME_VALUE (#PCDATA)>
<!ATTLIST INPUTNAME_VALUE Title CDATA "Internal Form Name:">
<!ELEMENT INPUTNAME_MAPPING (#PCDATA)>
<!ATTLIST INPUTNAME_MAPPING Title CDATA "Mapped UI Field Name:">
<!ELEMENT INPUTTYPE (#PCDATA)>
<!ATTLIST INPUTTYPE Title CDATA "Form Type:">
<!ELEMENT INPUTVALUE (DEFAULTVALUE*,OPTIONALVALUE*)>
<!ELEMENT DEFAULTVALUE (DEFAULTVALUE_DISPLAY, DEFAULTVALUE_INTERNAL,
MAPPING?)>
  <!ELEMENT DEFAULTVALUE_DISPLAY (#PCDATA)>
  <!ATTLIST DEFAULTVALUE_DISPLAY Title CDATA "Displayed Default
Value">
  <!ELEMENT DEFAULTVALUE_INTERNAL (#PCDATA)>
  <!ATTLIST DEFAULTVALUE_INTERNAL Title CDATA "Internal Default
Value">
  <!ELEMENT MAPPING (#PCDATA)>
  <!ATTLIST MAPPING Title CDATA "Internal Value MAPPING">
  <!ELEMENT OPTIONALVALUE (OPTIONALVALUE_DISPLAY,
OPTIONALVALUE_INTERNAL, MAPPING?)>
  <!ELEMENT OPTIONALVALUE_DISPLAY (#PCDATA)>
  <!ATTLIST OPTIONALVALUE_DISPLAY Title CDATA "Displayed Optional
Value">

```

Fig. 5.5. DLDL schema for DL search field description.

Based on the schema we can use the DLDL to depict details about a DL's search interface. In Figure 5.6 we provide part of the DL specification of the keyword feature as it occurs in the NEEDS digital library and how it maps to the UI. From the specification we know that NEEDS has a text input type search filter labeled "keywords" and its length is 35. It can be mapped to the UI_keyword search field of the unified interface.

```

<FORMFIELD>
  <REQUIRED Title="Required Field or not:">Y</REQUIRED>
  <WEIGHT Title="Weight of Field:">1</WEIGHT>
  <TYPE Title="Search Criteria/Display Option:">Search Criteria</TYPE>
  <LABEL Title="Displayed Field Name:">Keywords</LABEL>
  <LENGTH Title="Field Length:">35</LENGTH>
  <INPUTNAME>
    <INPUTNAME_VALUE Title="Internal Name:">keywords</INPUTNAME_VALUE>
    <INPUTNAME_MAPPING Title="Mapped UI Field
Name:">UI_keyword</INPUTNAME_MAPPING>
  </INPUTNAME>
  <INPUTTYPE Title="Form Type:">text input</INPUTTYPE>
  <INPUTVALUE />
</FORMFIELD>

```

Fig. 5.6. Part of DLDL specification for NEEDS.

Once we have the complete description of a digital library's search features, from its specification we can recreate or emulate its native interface. Figure 5.7 illustrates the emulated search interface of NEEDS generated from its specification.

Keywords	<input type="text"/>
Title	<input type="text"/>
Author/Creator	<input type="text"/>
Publisher	<input type="text"/>
Subject Heading	<input type="text"/>
Affiliates	Select an affiliate <input type="text"/>
Platform	Select a platform <input type="text"/>

Fig. 5.7. Emulated search interface for NEEDS based on specification.

The search filter information of a DL's specification is used to both generate the relevant part of the UI and, when filled in by the user, generate the queries issued to that DL. For example from the specification of NEEDS the mapping of the UI_keyword field of the LFDL UI is Keywords. Therefore, when a LFDL service users submits a query using the filter UI_keyword (e.g., UI_keyword= "computer"), the LFDL will translate the query to the native format of NEEDS (Keywords= "computer"). Table VI demonstrates a sample query translation from LFDL to NEEDS. Table VII gives some other DL's native query mapped to the same sample query in the LFDL.

TABLE VI
SAMPLE QUERY MAPPING BETWEEN NEEDS NATIVE QUERY AND LFDL UI

Sample Query in UI	UI_keyword=computer&UI_creator=Smith&UI_hits=20
Native Query after Mapping	keywords=computer&contributor=Smith&affiliates=&platform=&action=1&community=eng

TABLE VII
QUERY MAPPING TO OTHER DLS

DL	DL native query after mapping
ACM	query=computer&coll=ACM&dl=ACM&whichdl=acm
IEEE	rq=0&col=allieee&qt=computer&qc=allieee&nh=20&ws=0&qm=0&st=1&lk=1&rf=0&rq2=0
CogPrints	abstract/keywords/title=computer&abstract/keywords/title_srctype=ALL&authors/editors=Smith&authors/editors_srctype=ALL&_satisfyall=ALL
LTRS	abs=computer&au=Smith&sti=* &boolean=AND

5.2.4 Dynamic Interface Generation Algorithm

We have identified the factors affecting the dynamic generation of the interactive interface as: a set of keywords and the corresponding document hits of each keyword

within each DL; a base, generic universal search interface; and a complete, accurate description of each DL's search interface and query format. From the user input and keyword-hits information the most relevant DLs can be selected with a threshold algorithm tuned by user preference. But what features from each of these DLs should be included in the interface? A simple solution is to include all of those features but this produces an unacceptably complex search interface. Our algorithm considers the following factors: DL keyword relevance, an absolute filter weight from the universal interface, and a relative filter weight within each single DL. For example for keyword "network", DL A has 1000 hits while DL B has 300 hits. Therefore the features from A should have more weight than those from B. Each field from the universal interface has been given an absolute weight, e.g., the field UI_keyword is more important than the field UI_publisher. Also within a single DL some filters may play a more important role than others, so within that DL, filter 1 may have more weight than filter 2. One more factor is user search behaviors. In the LFDL a logging mechanism stores all user search interactions. By observing the log, if for a keyword, most of the times and most of the users apply a particular filter to place the search more weight will be given to that filter. The algorithm balances all those weights and selects those features with the highest weight and then presents them in the order of importance. The algorithm details are listed below.

- **Metrics**

- DL_j : digital library j
- $filter_i(DL_j)$: filter i of DL_j
- $filter_i(UI)$: filter i of the universal interface
- f_{UI} : overall absolute, static factor from the universal interface
- f_{hits} : overall dynamic factor from the relevance of the keyword input
- $W(filter_i(DL_j))$: relative weight of filter i within DL_j
- $W(filter_i(UI))$: relative weight of filter i within UI
- $\Sigma Hits(k)$: total of relevance hits from all DLs for a given keyword k
- $Hits(DL_j, k)$: number of hits or relevance from DL_j for a given keyword k
- $T_{UI}(filter_i)$: total calculated weight from UI for $filter_i$
- $T_{hits}(filter_i)$: total calculated weight from keyword relevance hits for $filter_i$

- $T(\text{filter}_i)$: total final calculated weight for filter_i
- TH : overall threshold to select a filter or not

- **Algorithm**

receive keyword k user input, get relevance hits from repository for a set of DLs that have records related to k ;

```

for (j = 1; j <= size of DLs set; j++) {
    get DL from setj;
    for (each filter  $i$  within  $DL_j$ ) {
        add  $\text{filter}_i$  to filters set  $s$ ;
         $T_{\text{hits}}(\text{filter}_i) += (\text{Hits}(DL_j, k) * W(\text{filter}_i(DL_j)))$ ;
    }
}
for (each filter  $l$  within filters set  $s$ ) {
     $T(\text{filter}_i) = W(\text{filter}_i(UI)) * f_{UI} * 100 + T_{\text{hits}}(\text{filter}_i) / \sum \text{Hits}(k) * f_{\text{hits}} * 100$ ;
    If ( $T(\text{filter}_i) > TH$ )
        Add  $\text{filter}_i$  to selected filters to be included in generated interface ;
}

```

- **Example**

Suppose the user submits a query on “computer” using the LFDL service which has ACM and NEEDS as participating DLs; each has relevant hits of 200 and 500 respectively for ‘computer’. The ACM native interface has a filter of CREATOR with a pre-defined weight of 1 and DATE with a weight of 0.5, while NEEDS has a filter of CREATOR with a weight of 1 and PUBLISHER with a weight of 0.2. Within the universal interface, the weights for CREATOR, DATE, and PUBLISHER are 80, 50, and 30. Also we give f_{UI} 0.3 and f_{hits} 0.7 and TH 50. The final overall calculated weight for each filter will be

$$T(\text{CREATOR}) = 80 * 0.3 + (200 * 1 + 500 * 1) / (200 + 500) * 0.7 * 100 = 94$$

$$T(\text{DATE}) = 50 * 0.3 + (200 * 0.5) / (200 + 500) * 0.7 * 100 = 25$$

$$T(\text{PUBLISHER}) = 30 * 0.3 + (500 * 0.2) / (200 + 500) * 0.7 * 100 = 20$$

Therefore, if we give a threshold of 50 only CREATOR will be included in the generated interface, and for a threshold of 20 all three filters will be selected.

We want to point out that though we log details about user's search behavior, that factor has not been implemented and not reflected in the algorithm yet.

The feature selection algorithm runs in real time as the user inputs keywords through the simple search mechanism. Since the time consuming "hit prediction" aspect of the algorithm runs off-line (therefore, not totally up-to-date), the performance is instantaneous. Once a user enters some or all of the presented fields, the queries should be well constrained to result in good precision (how good a result will depend on the user's effort). The final aspect of our quality of service promise is fast query results presentation. Being a distributed query system we do unfortunately depend on the participating DLs to respond quickly. We have implemented two features to increase performance: caching and immediate results display (or asynchronous results display: display the results as they come in from a DL instead of waiting for results returned from all DLs).

Figure 5.8 and Figure 5.9 show the different interfaces generated dynamically driven by different keywords as entered by a user. Note for demonstrational purposes we set the algorithm to show most of the filters of those related DLs. In the actual working version we need to fine-tune the threshold so that only the most relevant filters are presented to users. Unfortunately, for now the threshold setting is mostly from experience. It is desirable to have an algorithmic way to arrive at a meaningful overall threshold that is based on user preferences and overall access patterns to individual DLs. This has been left for future work.

For the keyword query "html", only the LTRS DL has hits, hence the dynamic interface resembles mostly the interface of LTRS, as shown in Figure 5.8. As for the query "university" in Figure 5.9, five DLs have related results with NEEDS and IEEE have larger number of hits. Therefore, more features and filters from NEEDS and IEEE will be included in the dynamic interface.

Figure 5.10 and 5.11 show the different generated interfaces for the same query "network security", given two different thresholds of 10 and 5. Obviously more filters are presented when threshold is 5 than when it is 10.

Home | Demo | About the Project | Bookmarks | Related Projects | Documents | Contact Us

Your search for "html" was found in these Digital Libraries

Below is the most commonly used interface for these DLs. Click [here](#) to customize and build your own search interface.

↔ Search specific bibliographic fields

keyword
 creator
 title
 whichfield AND OR
 id

↔ Display options

LIPS

search

Reports/articles
 Videos
 Audios
 Pictures

Fig. 5.8. Dynamically generated search interface for query "html".

Home | Demo | About the Project | Bookmarks | Related Projects | Documents | Contact Us

Your search for "university" was found in these Digital Libraries

Below is the most commonly used interface for these DLs. Click [here](#) to customize and build your own search interface.

↔ Search specific bibliographic fields

keyword
 creator
 title
 publisher
 hits
 source
 format
 Affiliates
 category
 date
 language
 whichfield

↔ Display options

COGRINTS NEEDS ILL MCR OTA

search

Reports/articles
 Videos
 Audios
 Pictures

Fig. 5.9. Dynamically generated search interface for query "university".

Your search for "network security" was found in these Digital Libraries

Below is the most commonly used interface for these DLs. Click [here](#) to customize and build your own search interface.

•► Search specific bibliographic fields

keyword	network security
hits	25
creator	
publisher	
title	

•► Display options

<input checked="" type="checkbox"/> COGPRINTS	<input checked="" type="checkbox"/> ACM	<input checked="" type="checkbox"/> NEEDS	<input checked="" type="checkbox"/> IEEE	<input checked="" type="checkbox"/> LTRS	<input checked="" type="checkbox"/> WCR
---	---	---	--	--	---

search

Reports/articles

Videos

Audios

Pictures

Fig. 5.10. Dynamically generated interface when threshold=10.

Your search for "network security" was found in these Digital Libraries

Below is the most commonly used interface for these DLs. Click [here](#) to customize and build your own search interface.

•► Search specific bibliographic fields

keyword	network security
hits	25
creator	
publisher	
title	
format	Select a platform ▼
date	2002

•► Display options

<input checked="" type="checkbox"/> COGPRINTS	<input checked="" type="checkbox"/> ACM	<input checked="" type="checkbox"/> NEEDS	<input checked="" type="checkbox"/> IEEE	<input checked="" type="checkbox"/> LTRS	<input checked="" type="checkbox"/> WCR
---	---	---	--	--	---

search

Reports/articles

Videos

Audios

Pictures

Fig. 5.11. Dynamically generated interface when threshold=5.

5.2.5 Additional User Customization Capability

We also want to provide the flexibility for a user to override the system generated interface by “hand picking” the fields she would like to see on the search interface. It has not been implemented yet but Table VIII and Figure 5.12 demonstrates the design.

First we present end-users a matrix of search interface features of the DLs in the LFDL federation. P denotes that a feature is utilized by a DL and it has predefined or DL-confined values, and F means it is a free word input or user can enter anything. For example, all of the DLs are using the search criteria of “title” and “creator”.

TABLE VIII
SEARCH FEATURES OF SELECTED DLS IN LFDL FEDERATION

			PROA	arXiv	NASA-CASI	ARC
search criteria	document metadata	title	F	F	F	F
		creator	F	F	F	F
		abstract	F	F	F	
		archive		P		P
		archive's set				P
		description				F
		type				P
		subject		F	F	
		language			F	P
		category			F	
		publisher			F	
		pub. date		P	F	F
		discover date				F
		full text	F			
		references	F			
		captions	F			
		others	PACS	Rep. Num; Journal ref; Comments	Rep. Num; Contract Num; Acc. Num; Journal meeting title; Corp. source	
	other filters	Fields conjunction		And, or, and not	And, or	And, or
others						
display options	results per page			P	P	
	sort order			P	P	
	others	Show errata				

From the above matrix users can get an idea what search criteria and options are supported by each DL so that they can use the features selection interface presented in Figure 5.12 to hand pick the features that they think are most suitable for their search needs.

Search Criteria	Document metadata	<input checked="" type="checkbox"/> Title	<input checked="" type="checkbox"/> Creator	<input checked="" type="checkbox"/> Abstract	<input checked="" type="checkbox"/> Archive
		<input type="checkbox"/> Archive's set	<input type="checkbox"/> Description	<input checked="" type="checkbox"/> Type	<input checked="" type="checkbox"/> Subject
		<input type="checkbox"/> Language	<input type="checkbox"/> Category	<input type="checkbox"/> Publisher	<input type="checkbox"/> Pub. Date
		<input type="checkbox"/> Discover date	<input type="checkbox"/> Full text	<input type="checkbox"/> References	<input type="checkbox"/> Captions
		<input type="checkbox"/> Others per DL			
Other filters	<input checked="" type="checkbox"/> Fields conjunction				
Display options	<input checked="" type="checkbox"/> Results/page	<input type="checkbox"/> Sort order			

Select sources:

<input checked="" type="checkbox"/> PROLA	<input checked="" type="checkbox"/> CASI	<input checked="" type="checkbox"/> arXiv	<input checked="" type="checkbox"/> Arc	<input checked="" type="checkbox"/> LANL
<input type="button" value="Submit"/>				

Fig. 5.12. User customization and search features selection interface.

5.3 EXPERIMENTATION AND DISCUSSION

There are currently few accepted ways to evaluate the effectiveness of DLs and their interoperability, compare different approaches, or to measure progress towards long-term goals [62]. The area of DL metrics is still quite young, but progress can be seen in the various white papers from the D-Lib Metrics Group [63], as summarized in [92], [98], and such sources as [1], [3], [32], [51] and some more general metrics related to Web performance [61]. Preliminary tests on the LFDL search service show that providing a federation service for non-cooperating digital libraries is feasible and that a dynamic,

user-centered interface is a practical approach to improve the quality of service, as well as service usability.

The objective of our experiments was to demonstrate that the LFDL provides a search service with satisfactory quality. To do this we calculate the accuracy of the LFDL by comparing the search results from the LFDL with those from accessing all DLs' native service in sequence using the same query. We simulate different search scenarios by submitting the following ten sample queries (not all DLs support all of the filters, we just tried using as many filters as possible) to each individual DL directly, and then to the LFDL:

- *Query 1: all about interoperability in Digital Library recently published in USA*
- *Query 2: an author in Stanford has a paper about freshness in Digital Library*
- *Query 3: all information about distance learning using internet (NEEDS)*
- *Query 4: a guy called Wilson from Johns Hopkins University just won NEEDS award for developing applet for signal processing courseware (NEEDS)*
- *Query 5: all about copyright of electronic or online publishing in recent 2 years (CogPrints)*
- *Query 6: all about intelligent agent (CogPrints)*
- *Query 7: the role of information technology in globalization process (CIAS)*
- *Query 8: all recent papers in aerodynamic (LTRS)*
- *Query 9: I have a dream by Martin Luther King (OTA)*
- *Query 10: all recent papers by Dr. K. Maly*

Table IX and Table X lists search results from each DL and LFDL, and also how accurate the LFDL is as compared with a native DL. For example for query 1 there are 44179 results from IEEE and 347 results from NEEDS. For the same query the LFDL returns 25 results each from IEEE and NEEDS (by default the LFDL search interface limits results from each DL to be 25, but users have option to change the limit), and those results are exactly matched with the top 25 results from IEEE or NEEDS directly.

TABLE IX
NUMBER OF DL NATIVE SEARCH RESULTS FOR EACH SAMPLE QUERY

	IEEE ¹	NEEDS ¹	CogPrints	CIAS	LTRS ¹	OTA
Q 1	44179	347			4	
Q 2	37138	347				
Q 3		46				
Q 4		1				
Q 5			6			
Q 6			10			
Q 7				1		
Q 8		1			227	
Q 9						1
Q 10					10	

¹ Match any of the keyword (for example, for “digital library” results returned for either “digital” or “library”)

TABLE X
NUMBER AND ACCURACY OF SEARCH RESULTS FROM LFDL FOR EACH DL¹

	IEEE	NEEDS	CogPrints	CIAS	LTRS	OTA
Q 1	25 (100%)	25 (100%)	0	0	4 (100%)	0
Q 2	25 (100%)	25 (100%)				
Q 3		25 (100%)				
Q 4		1 (100%)				
Q 5			6 (100%)			
Q 6			10 (100%)			
Q 7				1 (100%)		
Q 8		1 (100%)			25 (100%)	
Q 9						1 (100%)
Q 10					10 (100%)	

¹ For LFDL search we limit results from each DL to be 25, and an accuracy of 100% means the top 25 results are matched with the top 25 results of DL native search

The experiment shows that for the sample queries, the results returned by the LFDL are almost exactly matched with those from querying each individual DL directly. At this point we can demonstrate that the LFDL has satisfactory service quality. However, more experiments and evaluation are needed before we can declare that the LFDL has

achieved its objective completely. The current testbed is relative small. A document usually only exists in one DL but not in multiple DLs. Therefore, when we compose sample queries, for each query we have to aim toward one particular DL which may have reasonable results while other DLs may have nothing returned.

The user-centered, need-driven search interface is also more user-friendly and easy to use. Though it is not easy to design a quantitative way to measure system usability, we think the LFDL provides a better service usability as compared with other DLs which use traditional advanced search interfaces.

Query Routing

To provide efficient, highly useable federated search service across large scale, heterogeneous, distributed information sources, it is necessary to pick those most suitable for a give query. Query routing is the process to evaluate, select, and only distribute a query to the best, most relevant sources for that query [66], [117]. Unlike commercial web search engines which have a broad range of targets without limit on any topic, the LFDL is designed with the intention to server relatively small community concentrated on some given fields or topics. Therefore all DLs in the federation should be highly relevant to the field the federation serves. Still, query routing technique can improve the service and users' experience greatly if implemented properly. Currently we assume users are familiar with the DLs incorporated and give them choices of which DLs to search when they submit a query. We also trust the target DLs and include all results from them as long as users select those DLs. The following improvement can be done in the future:

- Currently the DLDL already support source information description by allowing each DL to disclose its metadata information like archival type and subjects/categories it serves in the XML specification. Such information can be used to evaluate which DLs will have the most relevant search results to a query.
- We already provide a dynamic interactive search interface based on each DL's keyword- hits information from local database. The same information can also be used to pick the most suitable DLs to be included in the distributed search.

- Based on a local search results repository (discussed in Section 6 and Section 7) we can do data warehousing or data mining on the results repository can sift out useful information about the nature and type of a DL that can be used for query routing.

SECTION 6

RESULTS PRESENTATION SERVICE: AUTOMATIC METADATA EXTRACTION

In the previous section we described the LFDL search service by presenting an interactive, user centered, need-driven advanced search mechanism based on Dublin Core metadata set.

In this section we introduce the LFDL results processing and presentation service, which collects and processes results from multiple DLs and then present the merged results to end users in a consistent way. We present an automatic metadata discovery and retrieval mechanism utilized by the service. The section is organized as follows:

- In section 6.1 we present an overview of the motive of the LFDL results processing and presentation service.
- We then in section 6.2 discuss the approach, design and implementation of automatic metadata extraction from non-cooperating DL search results.
- Finally, section 6.3 analyzes the initial experiences and discusses related work.

6.1 INTRODUCTION

The federated search service presented in the last section has a fairly high level of service quality in terms of precision/recall with rich functionalities for resource discovery. It demonstrates that providing a federation service for non-cooperating digital libraries is possible and that a dynamic user-centered search interface is a practical approach to improve the quality of service, as well as service usability.

However, so far all our work on the LFDL concentrated on fine-tuning the search, with little effort placed on processing the search results; they were presented in a flat structure. From interacting with individual digital libraries users are accustomed to seeing important information about a result record, such as the author identity, when and where it is published, and what it is really about (abstract, keywords, and/or subject). They may also want to manipulate the results in order to show only the results by a

particular author or after a particular date. All these require rich, interactive, and dynamic search result manipulation features. A straightforward way for presenting the result is to organize the results by DLs and for each DL list the titles of the hit along with links to show full records. Such service usability is not satisfactory, from the point of view of an end user. Organizing the result set helps users to locate the target object quickly in the result set. This requires post-processing of the result set, which is a challenging task in the distributed approach. Recall that the distributed search approach, in contrast to the harvesting approach, does not maintain the metadata from different collections locally. Ideally, if we can get all the metadata associated with the records in the search results, we could provide all of these services.

Performance is another major issue in a federated centralized service using distributed queries against non-cooperative DLs. In the LFDL rapid prototype system implementation, we improved the performance by using a local cache to store the query results. All results were cached according to the search query string so that if the same query were submitted, the local cache would be used instead of sending the query to a remote DL. However, such a cache mechanism was not flexible, efficient, and scalable. The cache reusability was low as only an exact matched query string resulted in a cache hit. For example the cache system would not know which field was a match for a particular query, author or publishing date. Records by author A, and records by author A published in year B will have two entries in the cache, which means considerable redundant information and a wasting of resources. Only a search against author A will hit the first entry and only a search against author A and year B will hit the second entry. Inefficiency also means less scalability. With too many redundant entries and limited available resources, such cache design cannot accommodate increasing number of queries and search results or if more DLs are included in a search. What one needs is a local repository with an “intelligent cache”, so a query on author A and year B will find entries in the cache, as already populated by an earlier query on author A. Intelligent cache means there are more cache hits without reducing the search result quality [111].

Both the tasks, organizing the result set for better service usability and intelligent caching, require additional processing of the result set using all the metadata available

from the result set. However, extracting metadata from a DL that is not cooperating is a non-trivial problem [10].

In this section we present an automatic metadata discovery and extraction mechanism based on the same principles we used to provide a search service to non-cooperative DLs: by observing the external behavior of a DL. The DLDL (Digital Library Definition Language) has been enhanced and an XML specification is used to define the rules to obtain metadata from each DL's result pages.

6.2 METADATA EXTRACTION FROM NON-COOPERATING DLS

In our approach a DL does not explicitly expose its metadata or how to obtain its metadata. Each DL has its own way to define metadata, and can display any subset of its metadata in whatever format at its own discretion. This makes it extremely difficult to post process search results to get metadata as there is no consistent way among DLs to expose them. In the following sub-sections we give details on challenges of extracting metadata from non-cooperating DLs and how we address them in the LFDL results presentation service.

6.2.1 Approach

In general each individual DL provides a search service by three web-based interfaces: an HTML form-based search page, a list of output pages of search results, and a detail page of a single record/document. In the LFDL we use a generic universal search interface based on Dublin Core elements, and we define each DL's behavior by using a specification that is generated based on each DL's search interface. The specification defines the rules of query mapping so that a federated search service can be provided. The results list page and/or document details page provides a possible source of result metadata. Typically, DLs list important meta information about each matched document on the search result page, and the metadata information matches closely to the Dublin Core metadata set. Even if no other meta information than the document title and a hyperlink to the document is available on the result page, more detailed meta information about a particular document or record will be presented once a user clicks on the

hyperlink. Therefore, an automatic metadata discovery and retrieval from a non-cooperating DL is possible as long as such metadata is reachable from its search results page and/or record details page. Our approach is to define rules on how to extract metadata from these pages, and to develop a metadata parser that will use these rules to obtain the metadata. As illustrated in Figure 6.1 the DL specification and DLDL have been extended to incorporate the extraction rules and the LFDL results process and presentation service will utilize the rules to parse metadata from DL result/record pages and then save extracted metadata to persistent local storage.

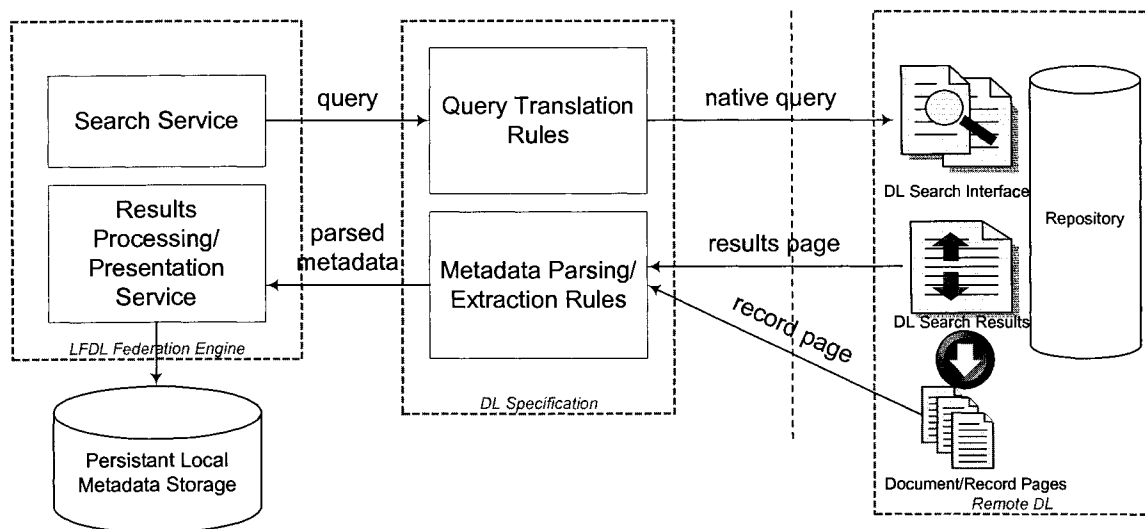


Fig. 6.1. LFDL metadata extraction approach.

Handling differences in metadata definition among different DLs is relatively easy. As we defined in the LFDL a generic universal search interface, we can use the Dublin Core metadata set as a common set, and all individual DL's metadata fields are mapped to the closest DC field. Hence, the LFDL search service will be based on DC fields. Some DLs may have fields that cannot be mapped to DC fields. We can define a set in addition to DC; if those fields are commonly used, we will map them to the extra set. If a

field is unique to a DL, we will still specify it and keep it. The metadata description of a DL will be limited to the exposed fields of that DL.

The difficult part is defining the rules to handle all the different cases of gathering metadata from search results and record pages of different DLs. Ideally, DLs would use consistent ways to make their metadata publicly available. For example all DLs could use the <meta> tag to display metadata information on their result and record pages, and they could all use the same DC element name as the <meta> name. If these are true, it would be straightforward in defining the parsing rules. Unfortunately, in reality each DL has its own way of displaying such meta information, and many times no meta tag is used but all information is in the actual HTML code. Therefore, our common metadata retrieval rules have to be generic enough to parse different result pages for different DLs.

TABLE XI
SAMPLE DL RESULTS AND METADATA DISPLAY PATTERNS

DL	Sample result (from results list page)	Metadata fields and display pattern
ACM	Becoming a computer scientist Amy Pearl , Martha E. Pollack , Eve Riskin , Elizabeth Wolf , Becky Thomas , Alice Wu Communications of the ACM November 1990 Volume 33 Issue 11 It is well known that women are significantly underrepresented in scientific fields in the United States...	title creator1, creator2 publication date description
NEEDS	The Knob & Switch Computer: A Computer Architecture Simulator for Introductory Computer Science (2001) Grant Braught; Computer Science Teaching Center Last Updated: 2002-02-01, Score: 336	title creator1, creator2; affiliation date, score
CogPrints	Barlow, Horace (1996) Intraneuronal information processing, directional selectivity and memory for spatio-temporal sequences.. Network: Computation in Neural Systems 7:251-259.	creator (date) title publication
CSTC	Integrating Empirical Methods into Computer Science Author: David Reed (daverreed@creighton.edu) Date: 05-05-2002 Category: Reviewed Demonstrations from Conferences Subject: Software - Programming Techniques	title creator1, creator2; date category subject
LTRS	1562 50 HZETRN: Description of a Free-Space Ion and Nucleon Transport and Shielding Comp	Title
NACA	885 27 Central automatic data processing system	Title
WCR	Analysis and modeling of World Wide Web traffic Conference Paper – G. Abdulla – Dept. of Computer Science, Va Polytechnic Institute and State University -- 1998	title type -- creator -- affiliation -- date

Table XI lists a few sample result pages to illustrate the differences among DLs. For example in Table XI the search results of the WCR digital library shows a document's title, type, creator(s), creator's affiliation, and publish date. The format of the metadata display is "title" followed by "type -- creator1, creator2... -- affiliation -- date". Despite the differences among DLs in displaying results, as long as within a given DL there is a consistent result displaying format or pattern, we can describe it for each DL so that the LFDL can process it accordingly.

6.2.2 Metadata Extraction and Parsing Process

We define DL output metadata at two levels: results list page level, and if available, record page level. Still, some DLs do not provide any metadata at all. Figure 6.2 illustrates the workflow of the Result Process Engine to retrieve and parse metadata from HTML pages at two levels.

- 1) Once search results (list page in HTML) from a DL arrive, the Result Process Engine checks for parsing rules from the DL's specification.
- 2) If metadata parsing rules have been defined and the results do have metadata included, the Process Engine applies parsing rules to get metadata from the result HTML page. It will then update the metadata cache with the extracted metadata.
- 3) If DL specification also defines lower level (record page level) metadata parsing rules, all record HTML pages will be retrieved from the remote DL, and the results will be parsed to get metadata as in step 2).
- 4) Extra process on cached metadata so that they are ready to be displayed.
- 5) After post-processing is done for all results from all DLs, results are merged and then displayed to end-users.
- 6) Periodically, cached metadata will be saved to persistent storage, in our implementation, a relational database.

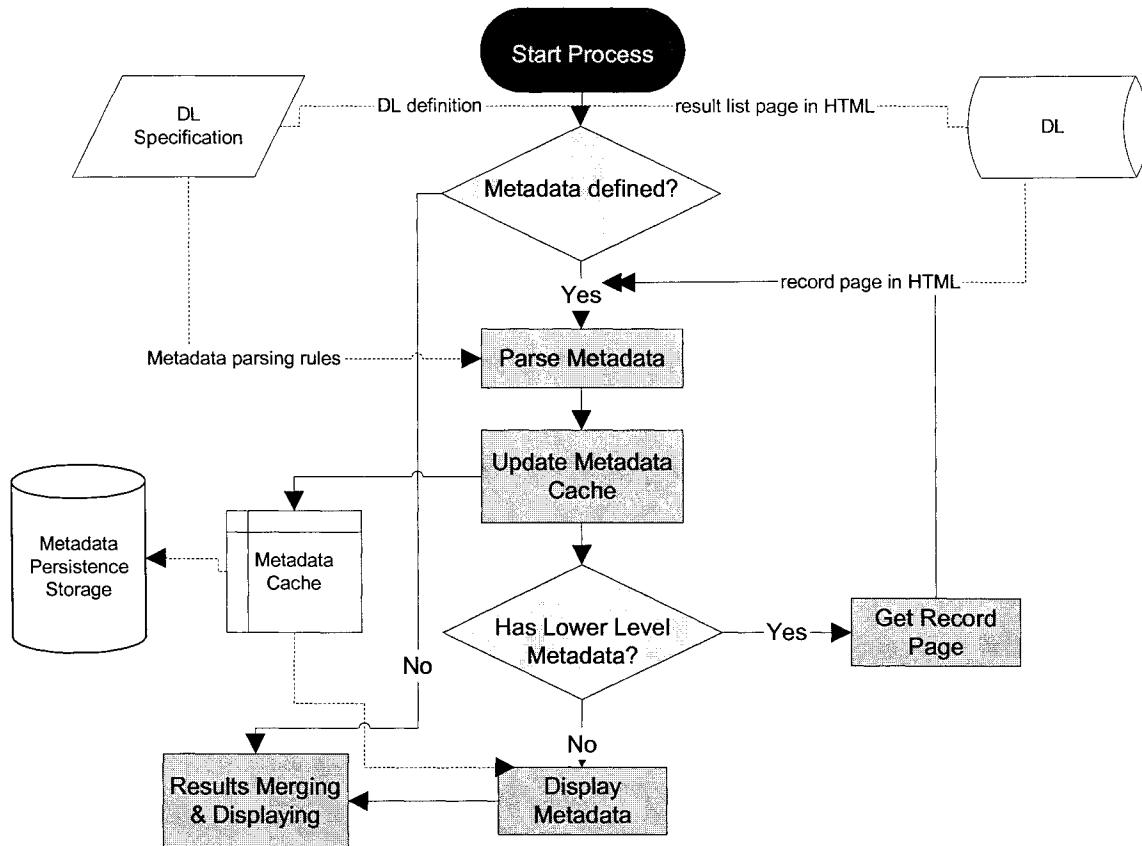


Fig. 6.2. Metadata retrieval and parsing workflow.

6.2.3 Metadata Parsing Rules Definition

We use the same DL XML specification to define metadata-parsing rules as we use for query mapping and metadata retrieval. We extend the DLDL to define parsing rules at two page levels: result list page level and single record document level. As shown in the DTD in Figure 6.3, the basic idea is that the raw string is separated into several segments, and each segment has one or several metadata fields. MATCH-START and MATCH-END specify a segment, and EXCLUDE and REPLACE will remove unrelated strings. Actual metadata fields will be separated by DELIMITER.

Figures 6.4 and Figure 6.5 show part of the XML specification of the ACM and Cogprint library based on the enhanced DTD for metadata parsing and extraction. For example the search results of ACM display a document's CREATOR field by beginning with `<div class="authors">` and ending with `</div>`. Therefore, when parsing the result

page, the LFDL process engine will parse and extract the content between those two strings as CREATOR. Similarly, Figure 6.5 demonstrates the metadata parsing rule for the CREATOR field of the record page of Cogrprints.

```

<!ELEMENT RESULT-METADATA (MATCH-START,MATCH-
END,EXCLUDE*,REPLACE*,DELIMITER*,METADATA-FIELD*) >
<!ELEMENT RECORD-METADATA (MATCH-START?,MATCH-
END?,EXCLUDE*,REPLACE*,DELIMITER*,METADATA-FIELD*) >
<!ELEMENT METADATA-FIELD (#PCDATA) >
<!ATTLIST METADATA-FIELD
    Title CDATA "information about a particular metadata
field">
<!ATTLIST METADATA-FIELD
    order CDATA #IMPLIED>
<!ATTLIST METADATA-FIELD
    multiple (true | false) #IMPLIED>
<!ATTLIST METADATA-FIELD
    delimiter CDATA #IMPLIED>
<!ATTLIST METADATA-FIELD
    format CDATA #IMPLIED>

```

Fig. 6.3. Part of DTD for DL parsing rule specification.

```

<RESULT-METADATA Title="Result page metadata parsing:"
hasRecordLevel="false">
    <MATCH-START enforced="true" Title="the beginning of matching
string of result metadata"><div class="authors"></MATCH-START>
    <MATCH-END enforced="true" Title="the end of matching string of
result metadata"></div></MATCH-END>
    <EXCLUDE Title="the string should be excluded or removed when
parsing"></EXCLUDE>
    <EXCLUDE Title="the string should be excluded or removed when
parsing"></EXCLUDE>
    <EXCLUDE Title="the string should be excluded or removed when
parsing"></EXCLUDE>
    <METADATA-FIELD order="1" multiple="true" delimiter="," "
Title="information about a particular metadata
field">CREATOR</METADATA-FIELD>
</RESULT-METADATA>

```

Fig. 6.4. Part of ACM DL specification for metadata parsing.

```

<RESULT-METADATA Title="Result page metadata parsing:"
hasRecordLevel="true">
  <MATCH-START Title="the beginning of matching string of result
metadata">null</MATCH-START>
  <MATCH-END Title="the end of matching string of result
metadata">null</MATCH-END>
</RESULT-METADATA>
<RECORD-METADATA Title="Record page metadata parsing:">
  <MATCH-START Title="the beginning of matching string of result
metadata">name="DC.title"</MATCH-START>
  <MATCH-END isLastIndex="true" Title="the end of matching string
of result metadata">" name="DC.creator"</MATCH-END>
  <EXCLUDE Title="the string should be excluded or removed when
parsing"/><meta content="</EXCLUDE>
  <REPLACE Title="replace old string with new string">
  <OLD-STRING Title="the old string to be replaced">"
name="DC.creator"</OLD-STRING>
  <NEW-STRING Title="replace with the new string">;</NEW-
STRING>
  </REPLACE>
  <METADATA-FIELD order="1" multiple="true" delimiter=";"
Title="information about a particular metadata
field">CREATOR</METADATA-FIELD>
</RECORD-METADATA>

```

Fig. 6.5. Part of DL specification for Cogprints.

6.3 EXPERIMENTATION AND RESULTS

We have implemented this architecture and created specifications for seven digital libraries (ACM, NEEDS, NACA, COGPRINTS, CSTC, LTRS, and WCR). All of these libraries are from the federation of the LFDL, therefore we only had to add the parsing and extraction rules to the DL specification documents. We illustrate the process for both a list page level DL and a record level DL. Figure 6.6 and Figure 6.7 show the form of metadata and how two very different DLs present them to the user. The ACM DL in Figure 6.6 displays a considerable amount of metadata information on the list page result, including TITLE, CREATOR, PUBLICATION, DATE, and DESCRIPTION. Earlier in Figure 6.4 we give a part of the specification that guides our engine in the extraction process.

However, as illustrated in Figure 6.7, Cogprints displays results on the list page using only the title metadata. The user has to click the title to obtain a record level page. Only that page has the metadata of interest. Again, Figure 6.5 shows part of the XML specification for the extraction process.

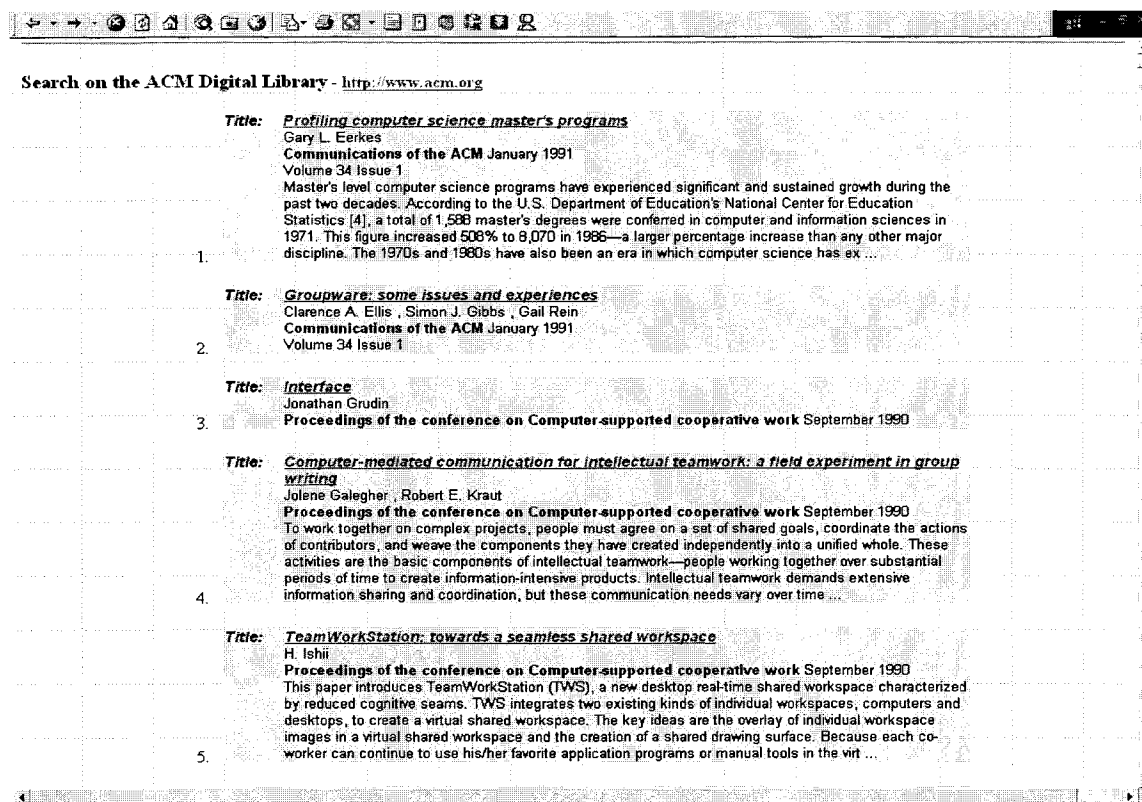


Fig. 6.6. Sample search results of ACM DL.

From the experience of adding the seven DLs to our federation we can say that on average the effort to observe and analyze a new DL is on the order of hours rather than days; these specific DLs took an average of three hours to define. This bodes well for the scalability of the approach at least from the specification perspective.

Finally, Figure 6.8 shows the results of our LFDL with metadata extraction. Both ACM and Cogprints appear as part of the LFDL results set in the same format and with metadata singled out. The amount of metadata will differ for each library and depends naturally on how much a library exposes in the result set.

Once metadata is parsed, it is stored in a local database to form a repository so that all future searches will be checked locally first before sending queries out to remote DLs. By using such a local repository, both search performance and service reliability will be improved. We call this “intelligent cache” as compared to the old caching mechanism in the LFDL prototype system.

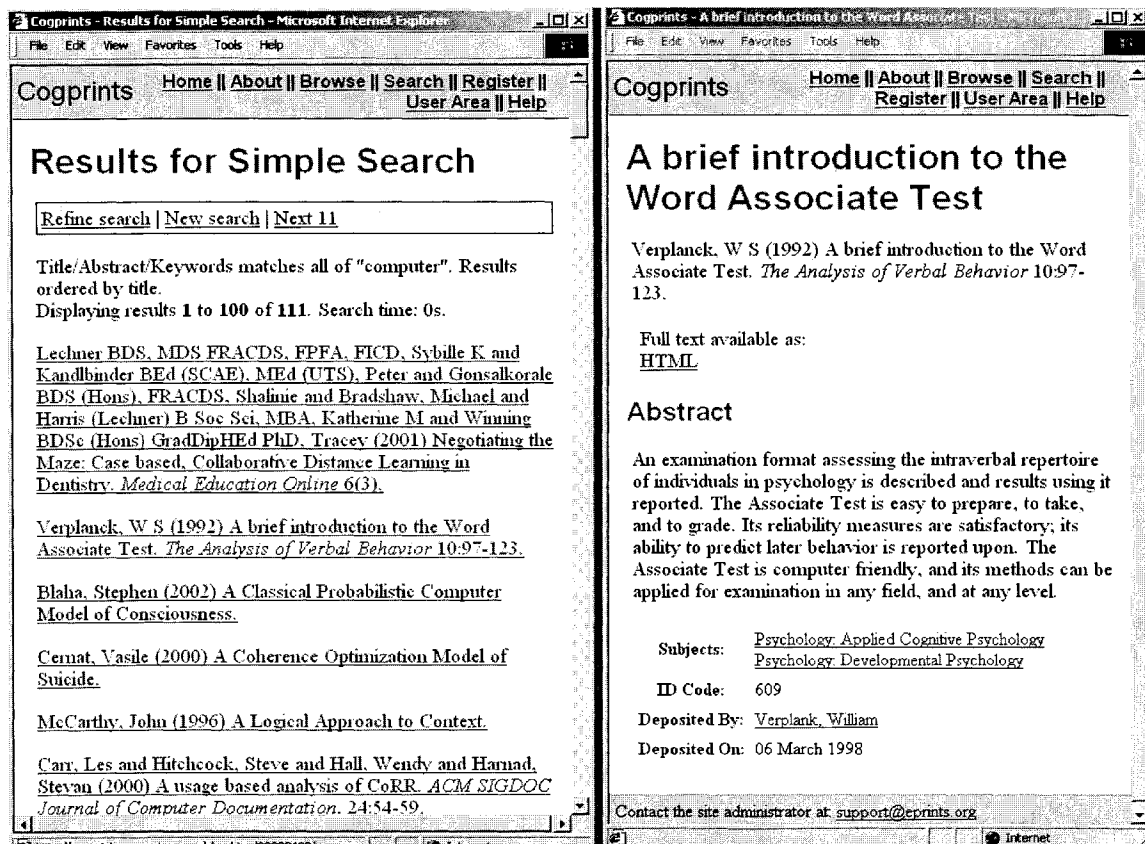


Fig. 6.7. Sample results list page and record page of Cogprints DL.

By using a cache grouped by metadata fields we can provide service at a quality as good as or close to the search service provided by an individual DL that maintains all the data it serves. A consistency engine will handle the cache consistency between local storage and remote DLs.

Metadata parsing and extraction is a resource intensive process and it may suffer scalability problems. Basically it uses string pattern matching from the raw HTML source code. Plenty of CPU time and memory are needed to process large number of HTML pages in short period, especially if those pages are large, which is not uncommon

in today's Web sites. In case each result page has dozens of links to particular records which also have metadata information, the LFDL will have to access each one of the linked document and extract metadata from it. For a common query for which each DL has plenty of hits, the LFDL may have to process hundreds or even more pages for one query. The response time may suffer when all these background processes are undergoing in real time.

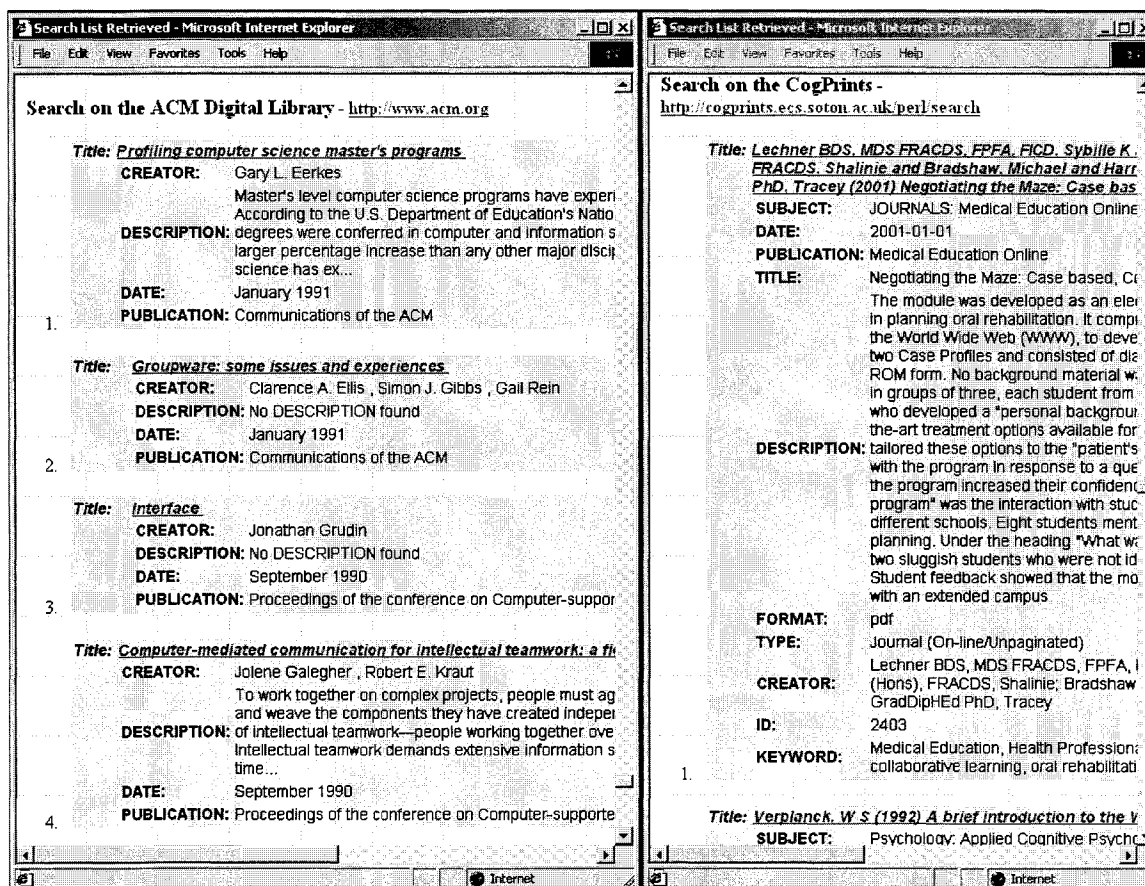


Fig. 6.8. Post processed results in LFDL after metadata parsing.

There are also cases that the LFDL cannot parse and extract metadata from certain HTML pages. It relies on certain patterns to parse raw strings and extract useful metadata information. Though rarely, the search result HTML source code of a DL may present metadata information in plain text without any particular pattern. In this case the LFDL

will not be able to handle the extraction. For example each record of a DL result page may display arbitrary number of metadata elements. Record 1 has “<TD>CREATOR A, CREATOR B, YEAR, PUBLISHER</TD>” while record 2 has “<TD> CREATOR, PUBLISHER</TD>”. The LFDL does not have enough information to distinguish different fields and do the extraction.

Results rank-merging

Currently the LFDL result presentation service is focus on processing DL native search results to fetch rich metadata. By default it displays results grouped by each DL and we have not addressed the result ranking/merging problem [34] which refers to processing and ranking search results from different source so that a federated service can merge and present them to end users in a meaningful way. It is also related to the query routing issue discussed in Section 5. This is a difficult task for the LFDL as all DLs in our federation do not reveal any of its internal structure including how it serves a query as well as its results ranking algorithm. On the one hand, for a given filter in a query we do not know if a DL uses exact match or fuzzy match to find results; and when there are multiple filters in one query we do not know if a DL uses AND/OR Boolean search. On the other hand, A DL may or may not disclose results’ rank information and even though a DL may display such information, it is only relative to the other documents in its own results set and does not represent an absolute measure of relevance for a query. Therefore we may be able to parse and process the ranking information of a DL’s results set, but without knowledge of the ranking algorithm it is hard to develop an effective methodology to compare ranking system of different DLs’ and then normalize and merge results together.

Another consideration is the tradeoff between results merging and performance. Instead of waiting for all results to be returned from all DLs, we display partial results whenever they are available from any DL. This improves system response time and also means merging results is not possible at least before the full results set is ready.

One possible exploration for future work is to assign a weight to each DL’s results set based on a DL’s overall relevance to a query (see query routing in Section 5), as well as if it uses a exact match or fuzzy match and the AND/OR Boolean on filters (such

information is possible to get though requires human intervene to study a DL results to some sample queries). Then we can design an algorithm to merge results from different DLs based on a document's relative weight as well as the weight of the DL that serves the result. Please also note that if there is a local copy of metadata from all DLs (discussed in Section 7), we can easily implement a ranking algorithm locally and present ranked, merged results to users.

SECTION 7

LOCAL REPOSITORY AND CACHING

In the previous section we introduced the LFDL results processing and presentation service by presenting an automatic metadata retrieval mechanism to extract metadata information from DL search results so that rich results can be presented to end users.

In this section we describe how the LFDL uses the retrieved metadata to build a local metadata repository. Based on the local repository we design and implement an intelligent cache to improve the performance and robustness of the federated service. The section is organized as follows:

- In section 7.1 we elaborate on the motive of building a repository from locally extracted metadata.
- In section 7.2 we discuss the approach, design and implementation of a local metadata repository.
- Section 7.3 describes how to utilize the locally maintained metadata in response to a search.
- In Section 7.4 we give details on the LFDL caching system based on the metadata repository.
- Finally section 7.5 analyzes the initial experiences and discusses related work.

7.1 INTRODUCTION

We described the LFDL results processing and presentation service in Section 6. To improve service usability we introduced an automatic metadata retrieval mechanism to explore deeper hidden web pages of non-cooperating DLs and provide rich search results from the extracted metadata.

In addition to improving usability locally obtained metadata also makes it possible to fulfill searches locally thus improving system performance and robustness. In this section we describe our efforts on building a local repository from extracted metadata and how we utilize this metadata repository to improve the LFDL federated service.

A local repository is common in the military or mobile computing community to provide a more reliable and efficient local information repository [89]. Instead of visiting each individual DL each time there is a search request, a reliable local server, which cached a local copy of resources provided by each DL, is accessed. This approach addresses the “information vulnerability” problem: dependence on dispersed/distributed information sources leaves us vulnerable to disruptions (loss of connectivity, information attacks), and limited bandwidth may preclude timely access.

In the LFDL the metadata are retrieved and stored in a local database to form a local federated repository to support future searches. We use a secondary level in-memory cache to improve the system performance further [112]. The added benefit of caching is that it allows processing the metadata (cached metadata) to lead to a quicker response time to a query and further it enables the exposure of the processed metadata through the OAI-PMH.

Though the LFDL uses a distributed search to achieve DL interoperability, by using a local metadata repository, it also takes advantage of the benefit of harvesting approach. Metadata are extracted from DLs without requiring each DL to follow any harvesting protocol. We can improve the performance of distributed search by checking local metadata first before sending the query to remote DLs. This way we are able to achieve the distributed approach’s lightweight interoperation among non-cooperating DLs with improved data freshness, and also we can benefit from the harvesting approach by providing quality service with better system performance and reliability. Once we have the metadata locally available, we can also improve system usability by supporting other richer services like locally records browsing.

7.2 LOCAL METADATA REPOSITORY

To make local metadata search really useful, the metadata repository has to be large enough for the search to find hits in sufficient numbers most of the time. The details of metadata retrieval and parsing are covered in Section 6. The metadata obtained from DL search results needs to be stored in the local repository after being retrieved and parsed. Over time and with larger numbers of different users performing queries this will lead to a varied repository.

We use an automatic fetching mechanism to create the repository, in addition to an active fetching agent based on the common keyword set we have. The first method is based on user search queries. Whenever there is a search request that cannot be fulfilled locally, the query is directed to remote DLs and metadata are extracted from the results returned from those DLs. Then the metadata can be stored in local repository. This is a passive method and depends on actual user interaction, and it is not enough to create a sizable local metadata repository in the beginning start-up phase.

Additionally, an intelligent agent or crawler can be used to actively visit each DL and fetch metadata from them. In Section 5 we already generate a common set of keywords that occur most frequently in a digital library's metadata records. The fetching agent can use keywords from that set to query the DL and thus extract metadata from the query results.

There are several issues with the agent approach. It is a heavy time- and resource-consuming process for both the harvester and target DLs. Also, though the queried keywords may be different, the query results may have many identical records. Therefore, considerable amount of redundant metadata parsing work has to be done. As a solution to the first issue, we can reduce the keywords sent to only those that are most frequently used in queries. Such information can be obtained from user search logs. For the second we can keep a parsed metadata list and once the agent detects that a result has been parsed, it stops parsing for that one and continues onto the next result. At this stage we have not implemented the two solutions yet and plan to leave them for future work.

The LFDL uses two levels of metadata storage: a permanent or persistent storage level and a transient or cache level. First, the metadata set obtained from DLs is stored in the local inventory. There are several options to implement such storage, such as database, plain text files, organized XML files, or data files in proprietary format. We use a traditional relational database instead of XML or other forms for better query efficiency, maintenance, and performance. Since we use Dublin Core as the basic metadata set, the database table structure for our metadata almost matches with the DC set.

Table XII shows the database table fields.

The fields that match the DC metadata elements are in the left column, and we defined some additional fields on the right column. For maintenance purposes each record was given an INTERNALID, DATELASTUPDATED, and STATUS. ARCHIVE is the DL from which the metadata was retrieved. DLs may define other metadata in addition to DC-compliant elements. Some general ones include CREATOR_AFFILIATION, KEYWORD, CATEGORY, PUBLICATION, and GROUPDATE. Also, a DL can store some meta information specific to itself using ADDITIONAL_FIELDS.

TABLE XII
STRUCTURE OF METADATA STORAGE TABLE

TITLE	INTERNALID
CREATOR	ARCHIVE
SUBJECT	ID_WITHIN_ARCHIVE
DESCRIPTION	CREATOR_AFFILIATION
PUBLISHER	PUBLICATION
CONTRIBUTOR	KEYWORD
DATESTAMP	CATEGORY
TYPE	GROUPDATE
FORMAT	DATELASTUPDATED
IDENTIFIER	ADDITIONAL_FIELDS
SOURCE	STATUS
LANGUAGE	
RELATION	

Keeping extracted metadata in a local database forms a reliable repository to provide the centralized, quick response search service. In order to achieve better system performance, we implement a secondary metadata storage by using in-memory cache within the search system. As illustrated in Figure 7.1, to serve a new query, the in-memory cache will be checked first instead of querying the database directly. Two-level caching makes it possible to provide a faster and more efficient search service.

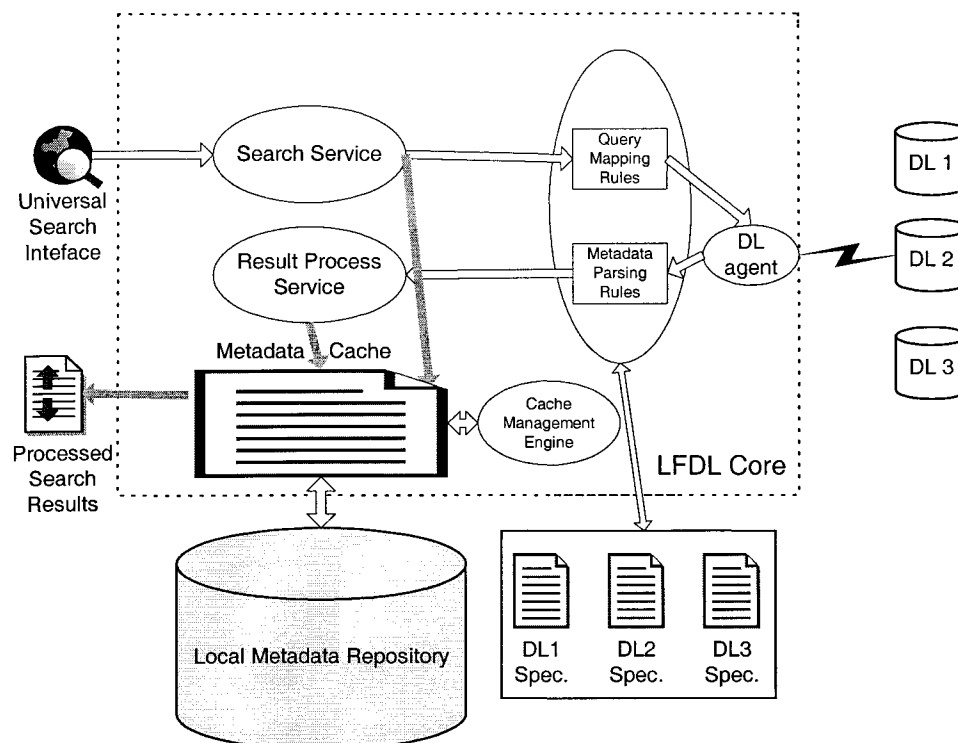


Fig. 7.1. LFDL metadata cache and repository.

7.3 LOCAL METADATA SEARCH

Since we are going to provide the search service locally, we have taken advantage of the relational database query language (SQL) to submit the query to the local metadata repository. When a user submits a search request using the LFDL unified search interface, the query string is translated to a SQL query and then sent to the database to get results from the metadata repository table. Here we use fuzzy string match, or use SQL language “LIKE” instead of “=”, to try to match each value between an HTML form field and the corresponding database table field, as both are based on the Dublin Core metadata element set. At the moment the LFDL only supports syntactical search; we do not parse the value of a given search field in the query string. For example to find all publications by an author with the last name Smith and the first name John published after June 2000, the fuzzy string match may return only documents with such database values as “CREATOR=John Smith” and “DATE=June 2000”, but not

“CREATOR=Smith, John” or “DATE=06/2000”. We can improve the search to handle the different formats or semantics of a filter or search field value, as the current metadata retrieval rules in XML already allow such semantic definition. For instance the XML DTD has a format attribute for each metadata field; we can define the format for CREATOR field as “Last Name, First Name” for DL A, and for DL B, “First Name Last Name”. And for the DATE field, DL A uses “mon date, year” while DL B uses “mo/da/yr”. Once such format definition is available, we can either convert the DL specific format to the unified LFDL format before storing values in our database, or use an application wrapper to convert the unified LFDL query string to a DL specific query and submit it to the local metadata database.

The LFDL prototype system presented the search results in a flat structure, which was not user friendly and the search usability was not appealing. Now that it is possible to get all the metadata associated with the records in the search results, we can provide an advanced user friendly search service with rich, customizable search results. Figure 7.2 shows the LFDL search interface, giving users a choice of displaying the results based on different grouping fields.

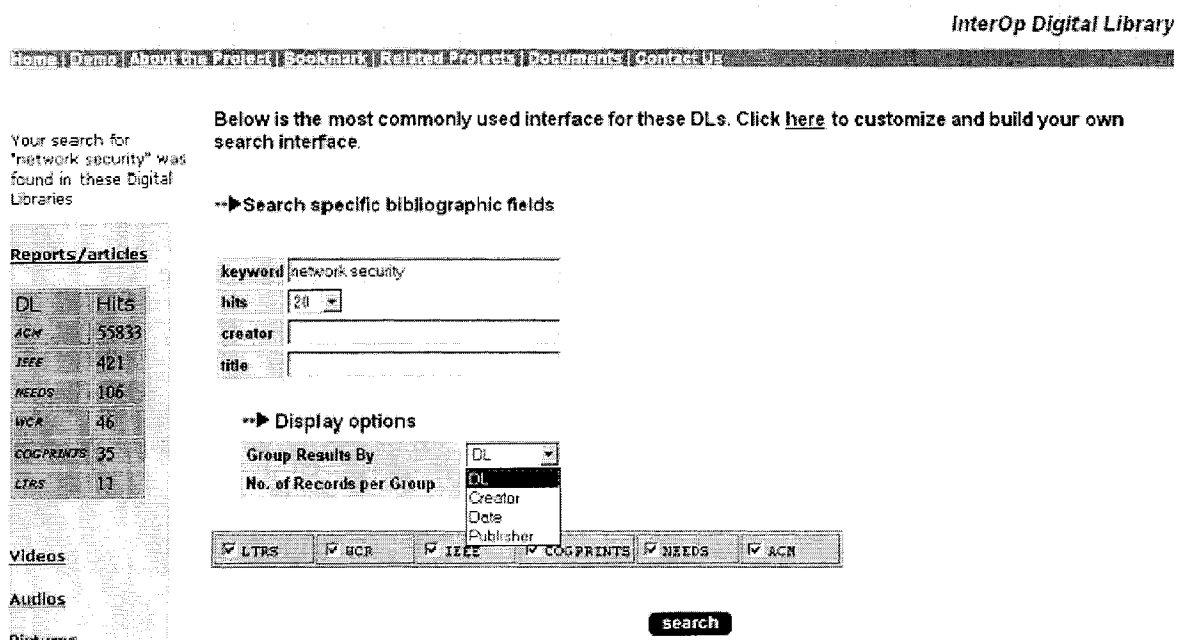


Fig. 7.2. LFDL interactive search interface.

The system can group results by any metadata element, but not all of the elements are useful to users. Here we demonstrate that a user can have the results displayed by each DL, Creator, Date, or Publisher.

Figures 7.3 and 7.4 display the results grouped by Date and Publisher respectively. Once inside the results page, a user can navigate the results without sending a new search request. We plan to use XML to format the results so that XSLT can be used to tailor the results to better serve user needs.

date: August 2003

TITLE:	<i>Ubiquitous computing security: Towards a new paradigm for securing wireless sensor networks</i>
ARCHIVE:	ACM
CREATOR:	K. Jones A. Wadaa S. Olariu L. Wilson M. Eltoweissy
DATE:	August 2003
PUBLICATION:	Proceedings of the 2003 workshop on New security paradigms
KEYWORD:	energy-efficient protocols frequency hopping security wireless sensor networks
DESCRIPTION:	The network model assumed in this paper consists of tiny, energy-constrained, commodity sensors massively deployed alongside with one or sink nodes that provide the interface to the outside world. The sensors in the network are initially anonymous and unaware of their location. The main contribution is to propose a new robust and energy-efficient solution for secure operation of wireless sensor networks. The paper motivates a new paradigm where security is based upon using parameterized fre...
PARSING METHOD:	Parsed Metadata
Date Last Accessed:	2004-05-28
Total Usage Counts:	118

1.

date: 01 Dec 02

TITLE:	<i>Network Security and Storage Security : Symmetries and Symmetry-Breaking</i>
ARCHIVE:	IEEE
DATE:	01 Dec 02
PUBLISHER:	IEEE Computer Society
DESCRIPTION:	December 11 - 11, 2002 Greenbelt, Maryland p. 3 Network Security and Storage Security: Symmetries and Symmetry-Breaking Donald Beaver Seagate
PARSING METHOD:	Parsed Metadata
Date Last Accessed:	2004-05-28
Total Usage Counts:	186

1.

TITLE:	<i>Network Security and Storage Security : Symmetries and Symmetry-Breaking</i>
ARCHIVE:	IEEE
DATE:	01 Dec 02
PUBLISHER:	IEEE Computer Society
DESCRIPTION:	xml version="1.0"?> Network Security and Storage Security: Symmetries and Symmetry-Breaking First International IEEE Security in Storage
PARSING METHOD:	Parsed Metadata
Date Last Accessed:	2004-05-28
Total Usage Counts:	182

2.

Fig. 7.3. Search results grouped by DATE.

publisher: Knowledgenet

1.	TITLE:	<i>Designing Microsoft® Windows 2000 Network Security : Windows 2000 Security</i>
	ARCHIVE:	IEEE
	DATE:	01 Aug 03
	PUBLISHER:	Knowledgenet
	DESCRIPTION:	This course provides the detailed technical knowledge necessary to design a security framework for small, medium, and enterprise networks .
	PARSING METHOD:	Parsed Metadata
	Date Last Accessed:	2004-05-28
	Total Usage Counts:	201
2.	TITLE:	<i>Designing Windows 2000 Network Security : Remote Access User Security</i>
	ARCHIVE:	IEEE
	DATE:	01 Aug 03
	PUBLISHER:	Knowledgenet
	DESCRIPTION:	This course provides the detailed technical knowledge necessary to design a security framework for small, medium, and enterprise networks using .
	PARSING METHOD:	Parsed Metadata
	Date Last Accessed:	2004-05-28
	Total Usage Counts:	159

publisher: Computer Science Teaching Center

1.	TITLE:	<i>CSC 475 Fall 1998 Syllabus</i>
	ARCHIVE:	NEEDS
	CREATOR:	Lawrence Rowe
	DATE:	1999
	PUBLISHER:	Computer Science Teaching Center
	KEYWORD:	Syllabus,multimedia,network,Internet,Web
	DESCRIPTION:	This syllabus describes a course covering networked multimedia issues, with an emphasis on the Internet
	MORE INFORMATION:	COURSEWARE SERIES:null
	PARSING METHOD:	Parsed Metadata
	Date Last Accessed:	2004-05-28
	Total Usage Counts:	202
	TITLE:	<i>XFS Demo (RW1-1)</i>
	ARCHIVE:	NEEDS
	CREATOR:	Mikram Narula
	DATE:	2001
	PUBLISHER:	Computer Science Teaching Center
	KEYWORD:	file,system,network,distributed
	DESCRIPTION:	Demo of key features of the xFS distributed file system.

Fig. 7.4. Search results grouped by PUBLISHER.

7.4 CACHING AND CACHE REPLACEMENT ALGORITHM

We use caching to make the LFDL system more robust and efficient, and also to provide a quicker or more responsive search performance. For the LFDL prototype system, we saved query string and query results in cache, so that when there is a new search request with the same query, the cache is read first, without visiting the remote DL. The key of the cache is a query string and the value is the results HTML page matching that query string. Obviously, this is not an efficient design. First, only when there was an exact match of the query string there would be cache hit. Second, the matching results to a query were unparsed and stored in cache as a whole html page, so there was too much redundant information and the cost of the system resources to store

and manage these values was significant. Now that we have parsed metadata available locally, we can implement a much more efficient “Intelligent Cache”. By intelligent cache we mean that a cache hit does not necessarily denote an exact query string match. Instead, the query string is translated into a more flexible SQL query and search against the local repository.

We implement a new caching mechanism using two levels of metadata storage: in memory cache and persistent database storage. The in-memory cache stores all recently used search results. The key is the internal ID of a metadata record, and the value is the metadata record itself. The metadata set from all DLs is stored in the local database. The search process consists of the following steps:

- 1) System starts, loads most recently and most often used metadata from database to memory cache.
- 2) User submits a query using the unified search interface.
- 3) Query is converted to local SQL query using predefined translation rules.
- 4) SQL query is sent to the local metadata database and the query results will be matching metadata internal IDs.
- 5) The memory cache is searched based on IDs, and if matched the metadata is merged; if not, the missing ones will be loaded from database to cache.
- 6) In the meantime, the original query string is transformed to a native non-cooperating DL query and sent to the remote DL. Results returned from the DL are parsed to extract metadata, which is saved to a local repository and loaded to the in-memory cache.

To better understand the caching mechanism, consider the following two search scenarios:

Case 1: a query for keyword=computer

Case 2: a query for keyword=computer AND date=2002

For our earlier caching design, assume query 1 and the results page have been cached; when query 2 is received, there will not be a cache hit as the query strings are different. Therefore, query 2 has to be sent out to remote DLs and then the results will be cached. Obviously results set 1 and results set 2 have a lot of records in common, but in

this case they have to be stored in cache separately as the results are not parsed and the common records could not be determined.

Under the new caching design, after query 1 is fulfilled, all results from the DL will be parsed and the metadata will be stored in the local repository, and then loaded into memory cache. For query 2 the local repository will be checked first and matching metadata IDs will be returned. Consequently, all matching metadata records will be found from cache by using those returned IDs. The only way that results are returned faster in the old design is when serving a repeated simple query that has been cached. In the old method results will be returned instantly while the new cache still has to query database to get matching metadata IDs first.

While serving requests from local repository and in-memory cache, the query is also sent to remote DLs in parallel and the results will be used to update local repository and cache so that any following request will have fresh data. Figure 7.5 shows some sample metadata records in the metadata cache.

For the implementation of the cache replacement algorithm based on the least used/least recently used (LRU, [118]) metadata records, we define the following metrics:

- **Initial System-wide Metrics**
 - *cache_max_size*: maximum number of metadata records allowed in cache
 - *cache_safe_size*: the number of records which are kept remain in cache when the cache is full and replacement algorithm is called to replace old records with new ones (this is to keep a just added item in cache from being replaced too soon even though its timestamp is new)
- **Runtime Cache Metrics**
 - *cache_size*: current number of metadata records in cache
- **Runtime Record-level Metrics**
 - *date_last_used*: the timestamp of when the record is last used
 - *total_usage*: the total number of times that the record has been used

When the LFDL system first starts, its cache is pre-populated using the following algorithm:

- **Cache pre-loading Algorithm**

System start, sort all metadata records in database based on date_last_used and total_usage;

while (cache_size < cache_max_size) {

load one metadata from sorted metadata queue to cache;

cache_size++;

}

sevemetadataatodb getmetadatafromdb clearmetadataacache

Metadata in Memory from DB

1.	TITLE:	<i>CSC 475 Fall 1998 Syllabus</i>
	ARCHIVE:	CSTC
	CREATOR:	Ronald Vetter
	DATE:	05-29-1999
	SUBJECT:	Computing Methodologies - Miscellaneous
	CATEGORY:	Syllabi and Reading Lists
	KEYWORD:	Syllabus multimedia network Internet Web
	DESCRIPTION:	This syllabus describes a course covering networked multimedia issues, with an emphasis on the Internet
	PARSING METHOD:	Parsed Metadata
2.	TITLE:	<i>CS294-1 Fall 1998 Reading List</i>
	ARCHIVE:	CSTC
	CREATOR:	Lawrence Rowe
	DATE:	05-29-1999
	SUBJECT:	Computer Applications - Miscellaneous
	CATEGORY:	Syllabi and Reading Lists
	KEYWORD:	Reading multimedia
	DESCRIPTION:	This reading list pertains to a multimedia course covering system architecture and development issues
	PARSING METHOD:	Parsed Metadata
	TITLE:	<i>EDSAC Emulator</i>
	ARCHIVE:	CSTC
	CREATOR:	Martin Campbell-Kelly
	DATE:	09-06-2001
	SUBJECT:	Computing Mileux - History of Computing
	CATEGORY:	Multimedia
	KEYWORD:	EDSAC Emulator
	DESCRIPTION:	The EDSAC was the world's first stored-program computer to operate a regular computing service. Designed and built at Cambridge Unive

Fig. 7.5. Sample metadata in LFDL metadata cache.

During normal system operation, all queries are checked locally from the cache and whenever there is a cache hit and an item is selected, its `date_last_used` and `total_usage` will be updated. In case of a cache miss the missing record will be loaded from the

database to the cache. If the cache has reached its maximum capacity, the newly loaded record will replace a current item in cache using the following algorithm:

- **Cache Replacement Algorithm**

sort all records in cache based on date_last_used;
keep those most recent used records, sort the remaining (cache_max_size - cache_safe_size) records based on total_usage;
save the least used record which has the lowest total_usage to database, and then replace it with the newly loaded record;

The algorithm here is a straightforward cache replacement implementation. For future improvement we can refine it and design a more sophisticated one, such as using a weight based solution which combines the factor of usage and timestamp.

7.5 EXPERIMENTATION, RESULTS, AND ANALYSIS

Caching plays a vital role in our approach. Studies in other fields have demonstrated that caching is an applicable approach in building efficient information retrieval systems. Pitkow [97] presents a simple, robust, adaptive caching algorithm for WWW-based information system. Markatos [78] reports on caching search engine results and shows that in the queries submitted to popular web search engines there exists a significant amount of locality: 20-30% of the queries have been previously submitted. Based on his simulation a medium-sized cache is enough to hold the results of most of the repeatedly submitted queries: a 300MB cache can achieve a hit rate of around 20%.

We have designed initial experiment and analyze results to test the effectiveness of the LFDL intelligent cache in terms of service response time. The objective was to demonstrate that it has better performance and response time than that of the earlier implementation of the LFDL cache with the simple mapping of query string and unparsed results. We use LFDL v2 for the version with improved caching and LFDL v1 for the LFDL with original cache design. The basic method was to submit a set of simulated search queries to each system and calculate the corresponding service response time. We used the following search scenario:

Case 1: a query for keyword=computer

Case 2: a query for keyword=computer AND date=2002

Case 3: a query for keyword= computer AND date=2002 AND creator=Richard

Case 4: a query for keyword=computer AND date=2002 AND creator=Richard AND publisher=University of Oregon

Table XIII shows the results for each system. We begin when the cache is empty for both systems, and for v2 there are no metadata records related to the query. For query 1 it took both v1 and v2 around 48 seconds to return the first 14 results, and then another 12 seconds to load the remaining 50 hits. Therefore, the total response time is 60 seconds to present the complete 64 results. The discrepancy comes from the different response times of each individual DL, and the LFDL displays partial results whenever they are available and then merges them to show the complete results set. For query 2 it took v1 14 seconds to show the first 34 results and a total of 25 seconds to show the complete 54 hits. It could not benefit from the cache, even though query 1 had been cached, because the query string is different. However, for LFDL v2, it could use the cache to perform a metadata based search; it took only one second to return 20 hits. The remaining results will come from remote DLs directly after 18 seconds. For query 3, LFDL v2 found only three records from the local metadata repository and the remaining records were from the distributed search among remote DLs. For query 4, there was no local hit and all results were from remote DLs.

TABLE XIII
RESPONSE TIME COMPARISONS LFDL V2 VS. V1

	V2	V1
Query 1	48sec(14hits) / 60sec(64hits)	48sec(14hits) / 60sec(64hits)
Query 2	1(20) / 18(63)	14(34) / 25(54)
Query 3	1(3) / 11(38)	16(38)
Query 4	11(22) / 20(38)	16(38)

Here are some comments on the results:

- 1) The DLs included in this study were IEEE, CogPrints, NEEDS, and CSTC. To limit the overall system processing time we excluded some DLs with longer response times, and only allow 20 results from each DL.
- 2) We used related queries and earlier queries formed a superset of later queries. This is just for demonstrational purposes to show the benefits of the LFDL v2 cache design. However, in the real world, related queries occur more often than totally unrelated queries.
- 3) Note the difference in results returned by v1 and v2. In v2 we implemented our own query mechanism against each metadata field, which may be different from the query used by a remote DL. For example for query “keyword=computer”, v2 may return records where either the TITLE field or the DESCRIPTION field contains “computer”. Here, we want to emphasize the system performance but not getting results as closely as possible from individual DLs, as in LFDL v2 we basically build our own digital library from harvested metadata. It may return results other than those from the original digital library. Is this against the goal of building a federated service for non-cooperating DLs? It is an interesting issue we have to explore. At least here we can see that LFDL v2 provides faster service than v1 does.
- 4) For query 3 and query 4 there were not many local search results hits, while each DL still returned quite a few results. This is because LFDL v2 strictly fulfills the search using the AND operator, while a native DL may not support the AND or OR operators, and, actually uses OR when providing services for searches with multi-field criteria.

SECTION 8

REGISTRATION SERVICE AND MANAGEMENT SERVICE

In the previous sections we focused on the key LFDL services from the perspectives of end users. In this section we describe the LFDL registration service for DL experts and the management service for LFDL system administrators. The section is organized as follows:

- In section 8.1 we introduce the registration service which allows a new DL to be added to the LFDL federation by registering its DLDL specification.
- We then in section 8.2 present the LFDL management service which facilitates the monitoring and maintenance tasks of the federation system.

8.1 REGISTRATION SERVICE

To provide the federated service among distributed, autonomous digital libraries, the service provider needs to be aware of the existence of a DL repository. The LFDL registration service allows a new DL to be added to the LFDL federation by registering its specification. The objective is to make the process dynamic and transparent to end users: to add a new DL, no code change is necessary and the newly joined DL shall be integrated to the federated search on the fly. Once added, users can start to search it in real time.

8.1.1 Approach

As illustrated in Figure 8.1, a DL expert creates the specification using the DLDL for a digital library and stores it in a centralized server. To add the DL to the LFDL federation, he can use the registration interface provided by the registration service to register the specification. A LFDL Specification Validator will enforce its validity by checking if the format of the specification follows the standard DLDL schema before the DL can be successfully registered. After validation the specification is parsed and the rules specified, including query mapping rules and result parsing rules, will be populated

to the LFDL rule engine and will be enforced by the LFDL to provide search service and results process and presentation service. All these are done dynamically so that once registered, a DL will be available for search.

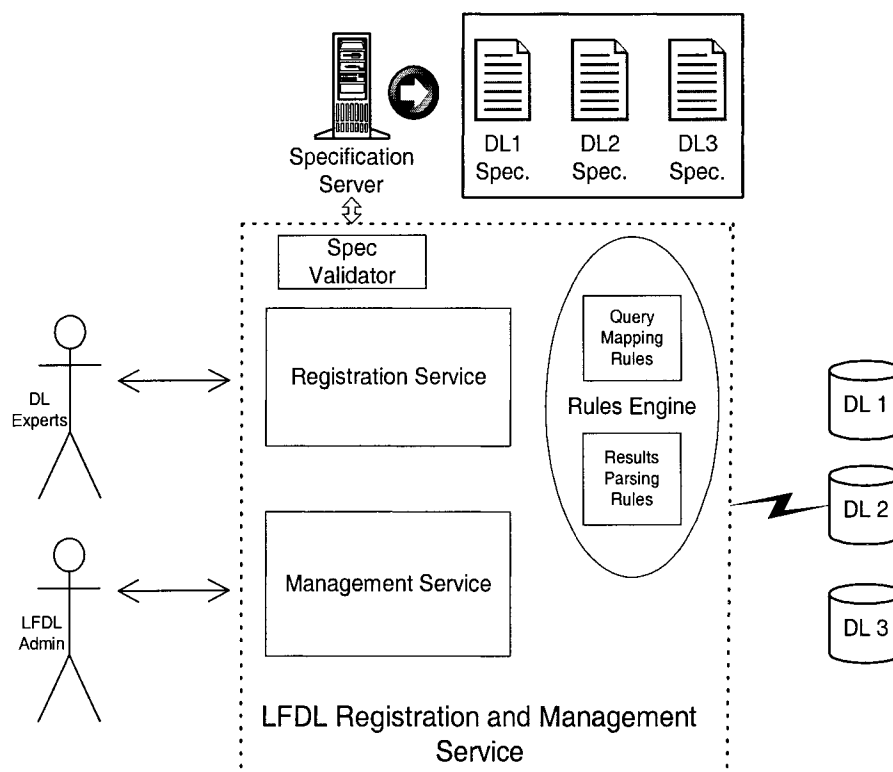


Fig. 8.1. LFDL registration and management service.

8.1.2. Design and Implementation

There are two approaches to implement the registry services: a separate LDAP [41], [121] based registry, and a tightly-integrated registration service. LDAP or Lightweight Directory Access Protocol is a specification for a client-server protocol to retrieve and manage directory information. The LDAP information model is based on the entry, which contains information about some object (e.g., a digital library). Entries are composed of attributes, which have a type and one or more values. Examples of attribute syntaxes are for strings, JPEG photographs, URLs and PGP keys [42].

We can implement the LDAP-based registration service by creating an entry for each registered DL, and each entry will have attributes such as DL description, category, and specification URL. We explored both implementation to evaluate their trade-offs.

LDAP Based Registration Service

In this approach, a digital library becomes part of a federated digital library by registering its description in DLDL to the LDAP server. For implementing this approach, we use the Netscape Directory Server 4.0 as an LDAPv3 server. The server held a master list of registered digital libraries and each individual DL had an entry which mapped the URL of its DLDL specification to its name. As an LDAP client the registration service was responsible for connecting to the LDAP server to retrieve or update the DL naming information and DL XML specification document information. JNDI (Java Naming and Directory Interface) API [46] was used to make the connection to the LDAP server and to access information from it. The JNDI API contains a naming interface (javax.naming) and a directory interface (javax.naming.directory). For this project we were using the naming interface as it provided the operations to do lookup on the LDAP server. To make the process more efficient, the LFDL cached the query results from the LDAP server. The cache results need to be refreshed only when there is an update or a new DL registration.

Figure 8.2 is borrowed from [132] and it illustrates the registration process.

- 1) Through a Web interface a DL expert sends a registration request to the LFDL registration service. The request consists of DL name and location of its specification.
- 2) The registration service verifies that the specification is valid and well-formatted following the DLDL schema.
- 3) Once validated, DL name and the URL of its specification will be saved to the LDAP server.
- 4) The registration result will be sent back to the user and displayed on his Web browser.

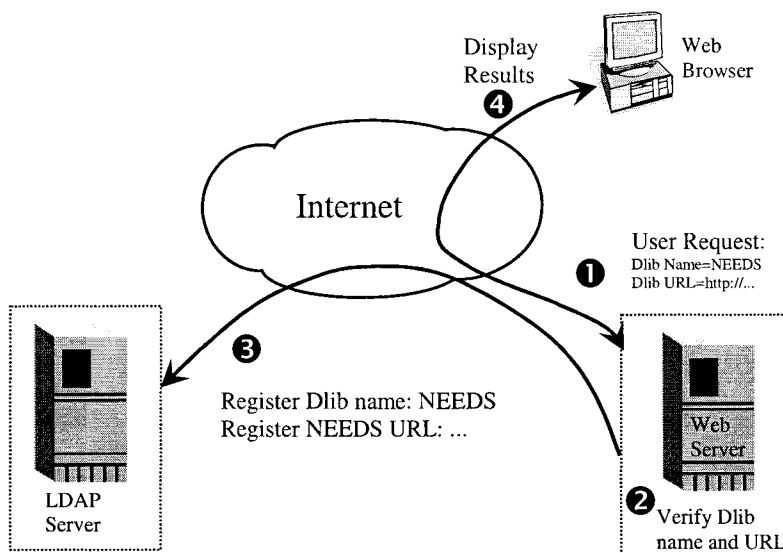


Fig. 8.2. LDAP-based registration process.

Tightly-Integrated Approach

Though the LDAP provides a standard, modular, and scalable solution to the LFDL registration service, it is not efficient, considering the nature of our registration requirement. After all, all we need is to save a DL's name and its DLDL URL. It is not necessary to go through an API call and an extra layer of storage to just access such simple and small amount of information. Therefore, in the current version we removed the LDAP layer and implemented a lightweight registration service.

In this approach we store DL and specification related information locally. Specifically, for each DL we store DL name and the URL of its DLDL specification in the local file system of the LFDL server. During normal operation, DL information mapping is kept in the LFDL server memory as a plain object. Table XIV displays sample name-value pairs stored in the in-memory map structure. All registration operations are fulfilled in memory. In case of server shutdown the information in memory will be serialized and saved to local disk which is available for read when server restarts. Alternatively, considering the registration requests are infrequent and serialization operation is virtually no cost, we could save the mapping information in memory to local storage whenever there is an update to avoid possible server crash.

TABLE XIV
REGISTRATION INFORMATION IN MEMORY

Name	Value
IEEE	Java URL object which holds the url of XML specification for IEEE
NEEDS	Java URL object which holds the url of XML specification for NEEDS

8.2 MANAGEMENT SERVICE

A well-managed information system can achieve desired functionality and improved performance and efficiency. For the LFDL we design and implement a monitoring and management service to facilitate such tasks. Figure 8.3 illustrates a Web interface that allows the LFDL managers to perform two sets of jobs: real-time system monitoring and run-time system reconfiguration.

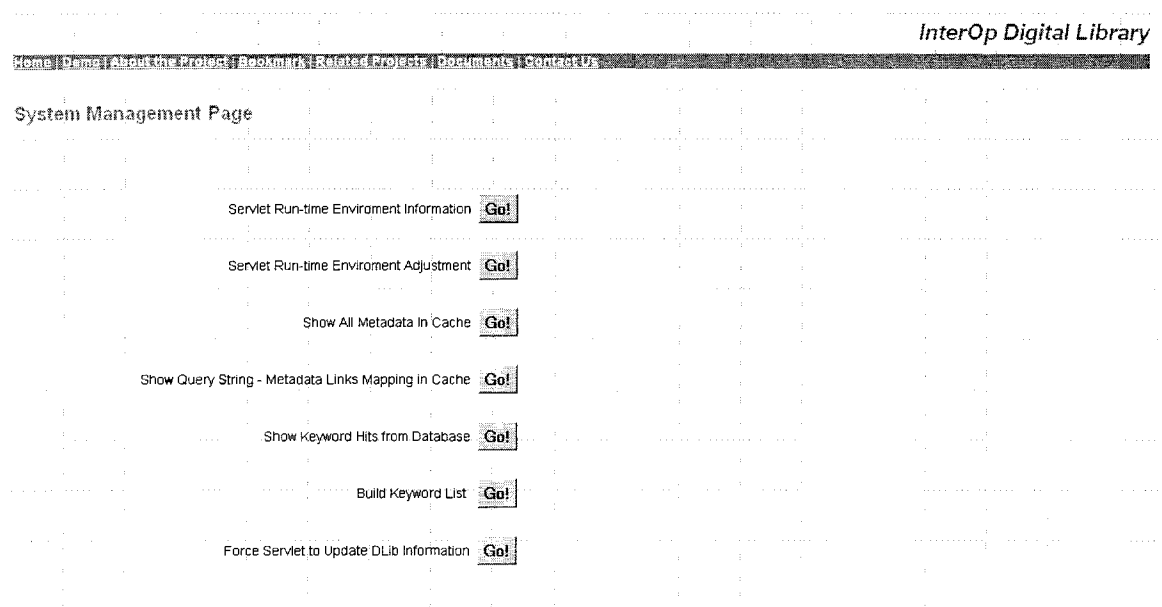


Fig. 8.3. LFDL management service interface.

8.2.1 Real-time System Monitoring

The LFDL manager can obtain informative real-time system information from the LFDL management web interface. The LFDL management service collects system runtime data so that the manager can monitor the system like tracking each DL's availability, average system response time, resource usage, and user search behavior data. Analyzing the statistical data helps to determine performance bottleneck and error prone points. Such information is critical for future system enhancement.

Table XV and XVI demonstrate two snapshots which displays various LFDL system runtime information and statistical data. Table XV displays the current version of the LFDL system, when it was started, how long it had been running, the memory usage, the total hits, the average system response time, and some other information.

TABLE XV
LFDL RUNTIME INFORMATION

Version	4.0.9
Program Start Time	Fri May 21 16:15:05 EDT 2004
Last Access Time	Thu Jun 03 16:38:06 EDT 2004
Up Time	13d 0h 28m 37s 727ms
Total Memory	27504640
Free Memory	2226912
Total Hits	78
Average Response Time (in ms)	1251
Metadata Cache Size	3000
Metadata Cache Keep Safe Size	100
Queries with results in Memory	26
Sum of result pages size	98599

Table XVI lists the current registered DLs and the URLs of their DLDL specification. There are also links to each DL's simulated search interface generated automatically from that DL's specification.

TABLE XVI
LFDL REGISTERED DL INFORMATION

ACM	http://www.cs.odu.edu/~shi/interop/demo/xmldoc/acm_082003.xml
IEEE	http://www.cs.odu.edu/~shi/interop/demo/xmldoc/ieee_122002.xml
NEEDS	http://www.cs.odu.edu/~shi/interop/demo/xmldoc/needs_100803.xml
COGPRINTS	http://www.cs.odu.edu/~shi/interop/demo/xmldoc/cogprints_063003.xml
CSTC	http://www.cs.odu.edu/~shi/interop/demo/xmldoc/cstc_122002.xml
LTRS	http://www.cs.odu.edu/~shi/interop/demo/xmldoc/ltrs_082803.xml
NACA	http://www.cs.odu.edu/~shi/interop/demo/xmldoc/naca_051903.xml
WCR	http://www.cs.odu.edu/~shi/interop/demo/xmldoc/wcr_090903.xml

8.2.2 Run-time System Reconfiguration

In addition to informative data displaying and real-time system monitoring, the LFDL management service also allows the system manager to fine-tune the system by adjusting runtime parameters, for example, allocating more memory. It would be an expensive job to restart an entire information system whenever there is a failed component during an execution, or just want to reconfigure system parameters [52]. Therefore, it is necessary to provide a run-time reconfiguration mechanism so that faulty component can be switched to alternate instances without affecting other parts of the system and not interfering end user services. Figure 8.4 shows some reconfiguration tasks available from the management interface. For example the system manager can turn on/off debugging mode, so that more or less system runtime information can be written to system logs. A detailed log facilitate pinpoint problems in case of there is system failure or other errors.

Action	Value
Toggle Debug Mode	No Value Needed
Update UI Threshold	
Update Max Metadata In Cache	
Update Metadata In Cache Keep Safe	

Fig. 8.4. LFDL reconfiguration utility.

SECTION 9

CONCLUSIONS AND FUTURE WORK

9.1 CONCLUSIONS

Digital library interoperability is essential in building federated services for end users to discover and utilize digital information from multiple sources through a single unified interface [67]. Creating such a service for existing heterogeneous DLs is the motivation of this work to build a lightweight federated service for libraries without prior coordination. This dissertation examines various approaches and answers the following questions by building the LFDL: Is it feasible to provide a realistic solution for interoperability among non-cooperating DLs? How do we create a lightweight, flexible, and efficient infrastructure to achieve such interoperability? How do we build the federated service to ensure satisfactory service quality, usability, system performance and reliability?

This research has successfully met the objectives as stated in Section 1. Our work on the LFDL system shows that it is possible to achieve interoperability among non-cooperating digital libraries and it is feasible to build an efficient, federated search service that works with non-cooperating digital libraries based on a distributed query approach. Dynamic, need-driven, and user-centered search is a practical approach to improve the quality and usability of service. Locally maintained metadata improves service usefulness and performance. The intelligent caching can further improve the service and achieve better efficiency. We created a test bed consisting of a dozen DLs and evaluated it against our objectives.

The following are the major contributions of our work on the LFDL in providing a federated service for non-cooperating digital libraries:

Scope

Digital Library interoperation has been an active research field in the DL community. However, most approaches require some level of cooperation among participating DLs. We think that there are still a number of DLs, like IEEE and ACM, which will continue

to work independently without participating in any interoperation, mostly for intellectual property concerns. This dissertation addresses the interoperation issues among non-cooperating DLs and presents a practical and efficient approach toward providing a federated search service for those DLs. A DL itself remains autonomous and joins the federation without making any changes to its library structure, data format, protocol and other internal features. The dissertation also provides an automatic metadata extraction mechanism, which has applicability beyond the objective of this thesis.

Architecture

The implementation of the LFDL is based on a lightweight, dynamic, data-centered and rule-driven architecture. To add a DL to the federation, all that is needed is observing a DL's interaction with the user and then storing the interaction information in a DL specification. The specification defines all interoperability processing rules and it is kept in a human-readable and highly maintainable format. The federation engine provides the federated service based on the specification of a DL. A registration service allows dynamically DL registration, removal, or modification. A federated service can be quickly formed for a special community; simply compose and register specifications of its DLs and those DLs will be incorporated into the service on the fly. Unlike other similar federation services, there is no hassle of code rewriting or recompiling just to add or change a DL. These notions are achieved by designing a new specification language in XML format (DLDL) and a powerful processing engine that enforces and implements the rules specified using the language. These techniques can be used in other application domains too, like a web robot [53], [54], a shopping agent and price comparison agent. Because of its many advantages over the traditional application architecture, Web Services is becoming a popular application solution among both industrial and research communities. The LFDL system fits well and can be easily adapt to a Web Services based infrastructure.

Approach

The most commonly used approach to achieve interoperability is one that harvests metadata into one central metadata repository that is then searched. One of its major

issues is the freshness of the data as this depends on the harvesting cycle. In this dissertation we explore an alternate approach where searches are distributed to participating DLs in real time. We have addressed the performance and reliability problems associated with other distributed search approaches. This is achieved by a locally maintained metadata repository extracted from DLs, as well as an efficient caching system based on the repository. In a sense the LFDL methodology lies in between the distributed search and the harvesting approach. Therefore, it has the former's advantage of data freshness and the latter's advantage of richer services, better performance and reliability.

Service Design

We also focus on service quality and usability. On the front end we introduced a dynamic user-centered, keyword driven search interface to improve service quality and usability. The same approach can be applied to other DL applications, like archon, to design a flexible interface based on archives and metadata. At the backend we provide an automatic metadata extraction mechanism to parse and process native DL search results so that the LFDL system can display rich results uniformly and consistently. Rich, processed search results further improves service usability and usefulness by providing enhanced search/navigation experience. Locally maintained metadata repository improves the LFDL caching system, and also makes it possible to provide additional high-level services. The automatic metadata parsing and retrieval can also be used by other domains and applications such as metadata extraction from PDF files. The intelligent cache further improves the performance, reliability, and efficiency of the LFDL system.

9.2 FUTURE WORK

We have demonstrated that it is feasible to build an efficient federated search service that works with non-cooperating digital libraries, yet, there are some issues that need to be addressed further. We will also briefly discuss some potential areas for future work.

Scalability

Scalability has been one of the biggest issues with the distributed search approach. It is not easy to incorporate a large number of new DLs at one time using the LFDL system. The cache size is not unlimited. Backend search results processing also affects system performance dramatically when too many DLs are included in the federated search and each DL has a large amount of results to be processed. Though in the LFDL we implement a not totally real-time result processing mechanism and try to keep end users transparent of the process, it remains quite resource consuming and may ultimately slow down the response time. More research is needed on the trade off between high quality service and better system performance. Still, the LFDL is useful for building services for special communities with a certain number of DLs.

On the other hand solely from the implementation perspective, it is possible that at some point a distributed search may have better performance over harvesting. In the harvesting approach a service provider has to have a huge metadata repository or database to accommodate metadata from all the participating DLs, and thus it is possible to make it slow to respond for queries if the service is not designed properly. While for a distributed search service provider could distribute the search burden among each individual DLs and just collect the results. And the asynchronous search utilized by the LFDL further addresses the network issue as well as various response times of different DLs. More experiments and evaluation are needed before we can assert which approach is definitely better.

DL Specification Generation

Currently, this is a manual process and requires some training and experience to learn the DLDDL and apply it when composing a specification for a DL. Human intervention is needed when a DL changes its searching and presenting schema. It will be beneficial to automate these processes so that both specification generation and DL behavior change discovery can be done automatically. One possible approach is to design a self learning system based machine learning on given examples [17], [24]. Another issue is that the LFDL can only support DLs with a standard HTML based interface and relatively simple web based interaction. For example, a DL with a Java applet based search, or a DL with

extensive user interaction and manipulation (e.g., a search requires multiple steps instead of one HTTP request and response) will be a problem. If necessary, we may extend the DLDDL and LFDL to include DLs with complex and non web-based search interfaces or other proprietary architecture and protocol.

Evaluation

The LFDL test bed is a relatively small DL set and we need to have more effective evaluation and measurement to test and assess the system usefulness, efficacy and service usability.

Implementation Issues

There are areas where we believe improvements will have potential payoffs. In the dynamic user interface generation, the keywords are chosen based on the static relevance of a DL without considering if it is really what the user wants. It is more reasonable to based on user selecting that DL and if the DL really has relevant results. As to the intelligent cache, one problem centers on populating the cache. Though we already have a basic keyword set and could use them to populate the cache, such process is very time consuming and produces redundant information. Similarly, it will take a long time to populate the cache through real users' searches in order to create a reasonably sized cache that will be helpful to users. We need to investigate trade-offs and other approaches. Maintaining the cache is another problem; for instance, what size is best considering resource efficiency and cache usage? How do we keep the cache consistent with remote DLs? A third problem concerns the intelligent caching of compound queries, typical query optimizations do not pose queries to a database when the first part of an "AND" query fails. Do we take into account such query optimizations for caching elements of compound queries?

Other possible enhancements include a personalized consumer portal, which is also suitable in the digital library community. We can customize the search interface and results displayed based on user searching behavior. We can also keep queries most often used by individuals and their other search preferences, like caching options either toward fresher data or faster results.

REFERENCES

- [1] P. S. Adler and M. M. Case. (1997) A national library for undergraduate science, mathematics, engineering, and technology education: A learning laboratory. [Online]. Available: <http://www.nap.edu/readingroom/books/dlibrary/appa.html#case>
- [2] H. Anan, X. Liu, K. Maly, M. Nelson, M. Zubair, J. C. French, E. Fox, and P. Shivakumar, "Preservation and Transition of NCSTRL Using an OAI-Based Architecture," in *Proc. of the Second ACM/IEEE Joint Conf. on Digital Libraries*, July 2002, pp. 181-182.
- [3] W. Y. Arms. (1997) A national library for undergraduate science, mathematics, engineering, and technology education: Needs, options, and feasibility (technical considerations). [Online]. Available: <http://www.nap.edu/readingroom/books/dlibrary/appa.html#arms>
- [4] W. Arms, *Digital libraries*. Cambridge, MA: MIT Press, 1999.
- [5] K. Arnold, J. Gosling, and D. Holmes, *the Java programming language, 3rd ed.* Addison-Wesley, 2000.
- [6] The Association of Research Libraries (ARL). (1995) Definition and purposes of a digital library. [Online]. Available: <http://www.arl.org/sunsite/definition.html>
- [7] D. Atkins. (1997) Report of the Santa Fe planning workshop on distributed knowledge work Environments. [Online]. Available: <http://www.si.umich.edu/SantaFe/report.html>
- [8] M. Baldonado, C. Chang, L. Gravano, and A. Paepcke, "The Stanford digital library metadata architecture," *International J. of Digital Libraries*, vol. 1, no. 2, 1997.
- [9] M. K. Bergman, "The deep web: Surfacing hidden value," *J. of Electronic Publishing*, vol. 7, no. 1, 2001.
- [10] D. Bergmark, "Automatic extraction of reference linking information from online documents," Tech. Rep. CSTR 2000-1821, Dept. of Computer Science, Cornell Univ., 2000.
- [11] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American*, vol. 284, no. 5, pp. 34-43, May 2001.
- [12] C. L. Borgman, "What are digital libraries? Competing visions," *Information Processing & Management*, vol. 38, no. 3, pp. 227-243, 1999.

- [13] C. M. Bowman, P. B. Danzig, and D. R. Hardy, "The Harvest information Discovery and Access System," *Computer Networks and ISDN Systems*, vol. 28, no. 1-2, pp. 119-125, 1995.
- [14] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, M. F. Schwartz, and D. P. Wessels, "Harvest: A Scalable, Customizable Discovery and Access System," Tech. Rep. CU-CS-732-94, Dept. of Computer Science, Univ. of Colorado, Aug. 1994.
- [15] M. Brown. (1996) FastCGI Specification. [Online]. Available: <http://www.fastcgi.com/devkit/doc/fcgi-spec.html>
- [16] CGI. [Online]. Available: <http://hoohoo.ncsa.uiuc.edu/cgi>
- [17] H. Chen, "Machine learning for information retrieval: neural networks, symbolic learning and genetic algorithms," *J. of the American Society for information Science*, vol. 46, no. 3, pp. 194-216, April 1995.
- [18] CogPrints. [Online]. Available: <http://cogprints.ecs.soton.ac.uk>
- [19] CORBA. Common Object Request Broker Architecture: Core Specification. [Online]. Available: <http://www.omg.org/docs/formal/04-03-12.pdf>
- [20] CSTC. [Online]. Available: <http://www.cstc.org>
- [21] R. Daniel and C. Lagoze, "Distributed Active Relationships in the Warwick Framework," in *Proc. of the 2nd IEEE Metadata Conf.*, Sept. 1997, pp. 16-17.
- [22] J. Davis and C. Lagoze, "NCSTRL: Design and Deployment of a Globally Distributed Digital Library," *J. of the American Society for information Science*, vol.51, no. 3, pp. 273-280, 2000.
- [23] J. Davis, D. B. Krafft, and C. Lagoze, "Dienst: Building a Production Tech. Rep. Server," in *Proc. of Advances in Digital Libraries*, 1995, pp. 211-222.
- [24] DEByE. [Online]. Available: <http://www.lbd.dcc.ufmg.br/~debye>
- [25] The Digital Library Research Group. [Online]. Available: <http://dlib.cs.odu.edu>
- [26] DL4U. [Online]. Available: <http://scholar.lib.vt.edu/DLI2>
- [27] The Dublin Core Metadata initiative. [Online]. Available: <http://dublincore.org>
- [28] N. Dushay, J. French, and C. Lagoze, "A characterization study of NCSTRL distributed searching," Tech. Rep. TR99-1725, Dept. of Computer Science, Cornell Univ., 1999.

- [29] S. Esler and M. Nelson, "Evolution of scientific and technical information dissemination," *J. of the American Society of information Science*, vol. 49, no. 1, pp. 82-91, 1998.
- [30] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. Dissertation, Univ. of California, Irvine, CA, 2000.
- [31] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "HTTP/1.1 -- Hypertext Transfer Protocol," RFC 2616, 1999.
- [32] N. Fortenberry. (1998) Report of the SMETE library workshop. [Online]. Available: <http://www.dlib.org/smete/public/report.html>
- [33] H. M. Gladney, E. A. Fox, Z. Ahmed, R. Ashany, N. J. Belkin, and M. Zemankova, "Digital library: gross structure and requirements: report from a March 1994 workshop," in *Proc. of the First Annual Conf. on the Theory and Practice of Digital Libraries*, June 1994, pp. 101-107.
- [34] L. Gravano, C. Chang, H. Garcia-Molina, and A. Paepcke, "STARTS: Stanford Proposal for internet Meta-Searching," in *Proc. of the 1997 ACM SIGMOD international Conf. on Management of Data*, 1997, pp. 207-218.
- [35] N. Green, P. G. Ipeirotis, and L. Gravano, "SDLIP + STARTS = SDARTS: A protocol and toolkit for metasearching," in *Proc. of the First ACM/IEEE Joint Conf. on Digital Libraries*, 2001, pp. 207-214.
- [36] S. M. Griffin, "Taking the initiative for Digital Libraries," *The Electronic Library*, vol. 16, no. 1, pp. 24-27, Feb. 1998.
- [37] S. M. Griffin, "Digital Library initiative - Phase 2," *D-Lib Magazine*, vol. 5, no. 7/8, 1999.
- [38] M. Hall. *Core Servlets and JavaServer Pages*. Prentice Hall, 2000.
- [39] J. Y. Halpern and C. Lagoze, "The computing research repository: promoting the rapid dissemination and archiving of computer science research," in *Proc. of the Fourth ACM Conf. on Digital Libraries*, 1999, pp. 3-11.
- [40] E. R. Harold and W. S. Means. *XML in a Nutshell*. O'Reilly, 2002.
- [41] J. Hodges and R. Morgan, "LDAPv3 technical specification," RFC 3377, 2002.
- [42] T. Howes, M. C. Smith, G. S. Good, T. A. Howes, and M. Smith. *Understanding and Deploying LDAP Directory Services*. Macmillan Technical Pub., 1998.

- [43] IEEE, institute of Electrical and Electronics Engineers. [Online]. Available: <http://www.ieee.org/>.
- [44] S. Iltis. (1995) Z39.50 - An Overview of Development and the Future. [Online]. Available: <http://www.cqs.washington.edu/~camel/z/z.html>
- [45] P. G. Ipeirotis, T. Barry, and L. Gravano, "Extending SDARTS: extracting metadata from web databases and interfacing with the open archives initiative," in *Proc. of ACM/IEEE Joint Conf. on Digital Libraries*, June 2002, pp. 162-170.
- [46] JNDI. [Online]. Available: <http://java.sun.com/products/jndi>
- [47] Java Servlet Technology. [Online]. Available: <http://java.sun.com/products/servlet>
- [48] B. Kahle, H. Morris, F. Davis, K. Tiene, and R. Palmer, "Wide area information servers: An executive information system for unstructured Files," *Electronic Networking: Research, Applications and Policy*, vol. 2, no. 1, pp. 59-68, 1992.
- [49] R. Kahn and R. Wilensky. A framework for distributed digital object services. Tech. Rep. cnri.dlib/tn95-01, CNRI, 1995. [Online]. Available: <http://www.cnri.reston.va.us/home/cstr/arch/k-w.html>
- [50] Kartoo. [Online]. Available: <http://www.kartoo.com>
- [51] J. H. Keller. (1997) Issues in developing a national library for undergraduate science, mathematics, engineering, and technology education. [Online]. Available: <http://www.nap.edu/readingroom/books/dlibrary/appa.html#kell>
- [52] I. Ko, K. Yao, and R. Neches, "Dynamic coordination of information management services for processing dynamic web content," in *Proc. of the eleventh international conference on World Wide Web*, 2002, pp. 355-365.
- [53] M. Koster. The Web Robots Page. [Online]. Available: <http://info.webcrawler.com/mak/projects/robots/robots.html>
- [54] M. Koster, "Robots in the Web: threat or treat?" *ConneXions - The interoperability Report*, vol. 9, no. 4, pp. 2-12, April 1995.
- [55] C. Lagoze and J. R. Davis, "Dienst - An architecture for distributed document libraries," *Communications of the ACM*, vol. 38, no. 4, pp. 47, April 1995.
- [56] C. Lagoze, W. Hoehn, D. Millman, W. Arms, S. Gan, D. Hillmann, D. Krafft, R. Marisa, J. Phipps, J. Saylor, C. Terrizzi, J. Allan, S. Lara, and T. Kalt, "Core services in the architecture of the National Science Digital Library (NSDL)," in *Proc. of the Second ACM/IEEE Joint Conf. on Digital Libraries*, 2002, pp. 201-209.

- [57] C. Lagoze, H. Van de Sompel, M. Nelson, and S. Warner. (2002) The Open Archives initiative Protocol for Metadata Harvesting, version 2.0. [Online]. Available: <http://www.openarchives.org/OAI/openarchivesprotocol.htm>
- [58] C. Lagoze and H. Van de Sompel, "The Open Archives initiative: Building a low-barrier interoperability framework," in *Proc. of the ACM/IEEE Joint Conf. on Digital Libraries*, 2001, pp. 54-62.
- [59] LTRS. [Online]. Available: <http://techreports.larc.nasa.gov/ltrs/ltrs.html>
- [60] S. Lawrence and C. L. Giles, "Searching the World Wide Web," *Science*, vol. 280, pp. 98-100, 1998.
- [61] B. Lavoie. (1999) Web characterization metrics. [Online]. Available: <http://www.oclc.org/research/projects/archive/wcp/default.htm>
- [62] B. Leiner. (1998) Digital library metrics workshop - goals. [Online]. Available: <http://www.dlib.org/metrics/public/6-98-workshop/goals.html>
- [63] B. Leiner. (1998) D-lib working group on digital library metrics. [Online]. Available: <http://www.dlib.org/metrics/public/metrics-home.html>
- [64] B. M. Leiner, "The NCSTRL Approach to Open Architecture for the Confederated Digital Library," *D-Lib Magazine*, vol. 4, no. 12, Dec. 1998.
- [65] Library of Congress. [Online]. Available: <http://www.loc.gov/z3950/gateway.html>
- [66] L. Liu, "Query Routing in Large-scale Digital Library Systems," in *Proc. of the international Conf. on Data Engineering (ICDE'99)*, March 1999, pp.154-163.
- [67] X. Liu, "Federating Heterogeneous Digital Libraries by metadata Harvesting," Ph.D. Dissertation, Dept. of Computer Science, Old Dominion Univ., Norfolk, VA, 2000.
- [68] X. Liu, K. Maly, and M. Zubair, "Enhanced Kepler framework for self archiving," in *Workshop on Distributed Computing Architectures for DLs*, 2002, pp. 455-461.
- [69] X. Liu, K. Maly, M. Zubair, and M. L. Nelson, "Arc: An OAI service provider for cross archive searching," in *Proc. of the ACM/IEEE Joint Conf. on Digital Libraries*, June 2001, pp. 65-66.
- [70] X. Liu, K. Maly, M. Zubair, and M. L. Nelson, "Arc - An OAI Service Provider for Digital Library Federation," *D-Lib Magazine*, vol. 7, no. 4, April 2001.
- [71] X. Liu, K. Maly, M. Zubair, and M. L. Nelson, "DP9 - an OAI gateway service for Web crawlers," in *Proc. of the Second ACM/IEEE Joint Conf. on Digital Libraries*, July 2002, pp. 283-284.

- [72] M. H. Maa, S. L. Esler, and M. Nelson, "Lyceum: A Multi-Protocol Digital Library Gateway," NASA Technical Memorandum 112871, July 1997. [Online]. Available: <http://techreports.larc.nasa.gov/ltrs/PDF/1997/tm/NASA-97-tm112871.pdf>
- [73] K. Maly, M. L. Nelson, and M. Zubair, "Smart Objects, Dumb Archives - A User-Centric, Layered Digital Library Framework," *D-Lib Magazine*, vol. 5, no. 3, March 1999. [Online]. Available: <http://www.dlib.org/dlib/march99/maly/03maly.html>.
- [74] K. Maly, M. Nelson, M. Zubair, S. Zeil, and X. Liu, "Structured Course Objects in a Digital Library," in *Proc. of the Third international Symposium on Digital Libraries*, 1999, pp. 89-96.
- [75] K. Maly, M. Zubair, H. Anan, D. Tan, and Y. Zhang, "Scalable Digital Libraries based on NCSTRL/Dienst," in *Proc. of the Fourth European Conf. on Digital Libraries*, 2000, pp. 169-179.
- [76] K. Maly, M. Zubair, and X. Liu, "Kepler - an OAI data/service provider for the individual," *D-Lib Magazine*, vol. 7, no. 4, 2001.
- [77] K. Maly, M. Zubair, M. L. Nelson, X. Liu, H. Anan, J. Gao, J. Tang, and Y. Zhao, "Archon - a digital library that federates physics collections," in *DC-2002: Metadata for e-Communities: Supporting Diversity and Convergence*, Oct. 2002.
- [78] E. P. Markatos, "On caching search engine results," Tech. Rep. 241, institute of Computer Science Foundation for Research & Technology - Hellas (FORTH), Greece, January 1999.
- [79] S. Melnik et al. Generic interoperability Framework. [Online]. Available: <http://www-diglib.stanford.edu/diglib/ginf/WD/ginf-overview>
- [80] MetaCrawler. [Online]. Available: <http://www.metacrawler.com>
- [81] NACA - National Advisory Committee for Aeronautics. [Online]. Available: <http://naca.larc.nasa.gov>
- [82] National Research Council. (1998) Report of a workshop, developing a digital national library for undergraduate science, mathematics, engineering, and technology education. [Online]. Available: <http://www.nap.edu/readingroom/books/dlibrary>
- [83] National Science Foundation. Digital Libraries initiative - Phase 2. [Online]. Available: <http://www.dli2.nsf.gov>
- [84] NEEDS - National Engineering Education Delivery System. [Online]. Available: <http://www.needs.org>

- [85] M. L. Neson, "Buckets: Smart Objects for Digital Libraries," Ph.D. Dissertation, Dept. of Computer Science, Old Dominion Univ., Norfolk, VA, 2000.
- [86] M. L. Nelson, K. Maly, and S. Shen, "Building a multi-discipline digital library through extending the Dienst protocol," in *Proc. of the Second international ACM Conf. on Digital Libraries*, July 1997, pp. 262-263.
- [87] M. L. Nelson, K. Maly, S. Shen, and M. Zubair, "Buckets: Aggregative, intelligent agents for publishing," *Webnet J.*, vol. 1, no. 1, pp. 58-66, 1999.
- [88] M. L. Nelson, K. Maly, S. N. T. Shen, and M. Zubair, "NCSTRL+: Adding multi-discipline and multi-genre support to the dienst protocol using clusters and buckets," in *Proc. of IEEE Advances in Digital Libraries 98*, April 1998, pp. 128-136.
- [89] NIMA. NIMA-in-a-Box debuts during Operation Allied Force, National Imagery and Mapping Agency. [Online]. Available: http://www.af.mil/news/Jun1999/n19990609_991144.html
- [90] NSDL, National SMETE Digital Library. [Online]. Available: <http://www.smete.org/nsdl>
- [91] NSDL Technical infrastructure white paper version 2.0. 2004. [Online]. Available: http://nsdl.comm.nsd.org/meeting/archives/smete/workgroups/technical/nsdl_tech_arch20.doc
- [92] NSF. (1999) Planning grant for the use of digital libraries in undergraduate learning in science. [Online]. Available: <http://www.nsf.gov/cgi-bin/showaward?award=9816026>
- [93] OAI, Open Archives initiatives. [Online]. Available: <http://www.openarchives.org>
- [94] OTA, Oxford Text Archive. [Online]. Available: <http://ota.ahds.ac.uk>
- [95] A. Paepcke, "Search middleware and the simple digital library interoperability protocol," *D-Lib Magazine*, vol. 6, no. 3, March 2000. [Online]. Available: <http://www.dlib.org/dlib/march00/paepcke/03paepcke.html>
- [96] A. Paepcke, C. K. Chang, T. Winograd, and H. Garcia-Molina, "Interoperability for digital libraries worldwide," *Communications of the ACM*, vol. 41, no. 4, pp. 33-43, April 1998.
- [97] J. E. Pitkow and M. M. Recker, "A Simple Yet Robust Caching Algorithm Based on Dynamic Access Patterns," in *Proc. of the 2nd World Wide Web Conf.*, 1994. [Online]. Available: <http://www.ncsa.uiuc.edu/SDG/IT94/Proc./DDay/pitkow/caching.html>

- [98] Planning Grant for the Use of Digital Libraries in Undergraduate Learning in Science: Evaluation Baseline. [Online]. Available: <http://dlib.cs.odu.edu/nsf/dlib2/udlfplan>
- [99] J. Powell and E. A. Fox, "Multilingual federated searching across heterogeneous collections," *D-Lib Magazine*, vol. 5, no. 8, Sep. 1998. [Online]. Available: <http://www.dlib.org/dlib/september98/powell/09powell.html>
- [100] A. L. Powell and J. C. French, "Growth and Server Availability of the NCSTRL Digital Library," in *Proc. of 5th ACM Conf. on Digital Libraries*, June 2000, pp. 264-265.
- [101] J. E. Refsnes. XML DTD - An introduction to XML Document Type Definitions. [Online]. Available: <http://www.xmlfiles.com/dtd>
- [102] Resource Description Framework (RDF). [Online]. Available: <http://www.w3.org/RDF>
- [103] G. Salton and C. Yang, "On the specification of term values in automatic indexing," *J. of Documentation*, vol. 29, pp. 351-372, 1973.
- [104] Santa Fe Convention. [Online]. Available: http://www.openarchives.org/sfc/sfc_entry.htm
- [105] M. Schwartz. (1996) Report of W3C distributed indexing and searching workshop. [Online]. Available: <http://www.w3.org/Search/9605-indexing-Workshop>
- [106] Search.com. [Online]. Available: <http://www.search.com>
- [107] SearchLight. [Online]. Available: <http://searchlight.cdlib.org>
- [108] E. Selberg and O. Etzioni. (1995) Multi-Service Search and Comparison Using the MetaCrawler. [Online]. Available: <http://www.2-sir.com/TwinFalls/metacrawler.html>
- [109] R. Shi, K. Maly and M. Zubair, "Interoperable federated digital library using XML and LDAP," in *Proc. of Global Digital Library Development in the New Millennium*, May 2001, pp. 277-286.
- [110] R. Shi, K. Maly and M. Zubair, "Dynamic interoperation of non-cooperating digital libraries," in *Proc. of international Conf. on Digital Library - IT Opportunities and Challenges in the New Millennium*, Beijing, China, July 2002.

- [111] R. Shi, K. Maly and M. Zubair, "Automatic metadata discovery from non-cooperative digital libraries," in *Proc. of IADIS international Conf. on e-Society 2003*, Lisbon, Portugal, June 2003, pp. 735-739.
- [112] R. Shi, K. Maly and M. Zubair, "Improving federated service for non-cooperating digital libraries," in *Proc. of international Conf. on Digital Libraries*, New Delhi, India, Feb. 2004.
- [113] H. Van de Sompel and C Lagoze, "The Santa Fe Convention of the Open Archives initiative," *D-Lib Magazine*, vol. 6, no. 2, Feb. 2000.
- [114] H. Van de Sompel and P. Hochstenbach, "Reference linking in a hybrid library environment Part 1: frameworks for linking," *D-Lib Magazine*, vol. 5, no. 4, April 1999.
- [115] H. van De Sompel, T. Krichel, M. L. Nelson, P. Hochstenbach, V. M. Lyapunov, K. Maly, M. Zubair, M. Kholief, X. Liu, and H. O'Connel, "The UPS prototype project: team, goals, motivation and relation to the Santa Fe convention," *D-Lib Magazine*, vol. 6, no. 2, Feb. 2000. [Online]. Available: <http://www.dlib.org/dlib/february00/vandesompel-ups/02vandesompel-ups.html>
- [116] Standard Generalized Markup Language (SGML). [Online]. Available: <http://www.w3.org/MarkUp/SGML>
- [117] A. Sugiura and O. EtzioniQuery, "Routing for Web Search Engines: Architecture and Experiments," in *Proc. of the Ninth international World Wide Web Conf.*, May 2000. [Online]. Available: <http://www9.org/w9cdrom/139/139.html>
- [118] A. S. Tanenbaum. *Modern Operating Systems*. pp. 218. Prentice-Hall, 1992.
- [119] H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. (2001) XML Schema 1.1. Tech. Rep. [Online]. Available: <http://www.w3.org/TR/xmlschema-0>
- [120] UDDI, the Universal Description, Discovery and integration. [Online]. Available: <http://www.uddi.org>
- [121] M. Wahl, T. Howes, and S. Kille, "Lightweight Directory Access Protocol v3," RFC 2251, Dec. 1997. [Online]. Available: <http://www.ietf.org/rfc/rfc2251.txt>
- [122] Web Services. [Online]. Available: <http://www.w3.org/2002/ws>
- [123] S. Weibel, "The Dublin Core: a simple content description model for electronic resources," *Bulletin of the American Society for information Science*, vol. 24, no. 1, pp. 9-11, 1997.

- [124] The World Wide Web Consortium (W3C). [Online]. Available:
<http://www.w3.org>
- [125] Extensible Markup Language (XML). [Online]. Available:
<http://www.w3.org/XML>
- [126] XML Schema. [Online]. Available: <http://www.w3.org/XML/Schema>
- [127] N. Ward, A. Wood, S. Finnigan and R. Iannella. (1996) Discussion Paper: Networked information Retrieval Standards. [Online]. Available:
http://www.dstc.edu.au/RDU/publications/html_reports/webir.html
- [128] WCR, Web Characterization Repository. [Online]. Available:
<http://repository.cs.vt.edu>
- [129] Z3950 Specification. Information Retrieval (Z39.50): Application Service Definition and Protocol Specification. [Online]. Available:
<http://www.loc.gov/z3950/agency/markup/markup.html>
- [130] M. Zubair, K. Maly and I. Ameerally, "XML-Based integration of heterogeneous digital libraries," Tech. Rep. TR- 99-06, Dept. of Computer Science, Old Dominion Univ., Norfolk, VA, Oct. 1999.
- [131] M. Zubair, K. Maly, H. Anan, D. Tan and Y. Zhang, "Scalable digital libraries based on NCSTRL/DIENST," in *Proc. of the European Conf. on Digital Libraries*, Sept. 2000, pp. 168-180.
- [132] M. Zubair, K.Maly, and R. Shi, "Focus research libraries in support of active learning," in *Proc. of international Conf. on information and Communication Technologies for Education*, Vienna, Austria, Dec. 2000.
- [133] M. Zubair, K. Maly, I. Ameerally, and M. Nelson, "Dynamic construction of federated digital libraries," in *Proc. of WWW9 Conf.*, May 2000, pp. 56-57.

APPENDIX A

REGISTERED DIGITAL LIBRARIES IN THE LFDL TEST BED

Name	URL	Organization	Archival Type	Subject
ACM	Portal.acm.org	Association for Computing Machinery	Journals, conference papers, Proc., news letters, transactions	Computer Science
COGPRINTS	Cogprints.ecs.soton.ac.uk	School of Electronics and Computer Science, Univ. of Southampton	Journals, conference papers, Proc., technical reports, book chapters	Biology, Computer Science, Linguistics, Neuroscience, Philosophy, Psychology
CSTC (Computer Science Teaching Center)	www.cstc.org	Computer Science Teaching Center	Conf. papers, lectures, multimedia materials	Computer Science
IEEE	www.ieee.org	institute of Electrical and Electronics Engineers	Journals, conference papers, Proc., news letters, transactions, web pp.	Computer science, engineering
LTRS (Langley Technical Reports Server)	techreports.larc.nasa.gov/ltrs	National Aeronautics and Space Administration	Journals, conference papers, Proc., technical reports	Aeronautics, computer science, physics, space science
NACA (National Advisory Committee for Aeronautics)	naca.larc.nasa.gov	National Aeronautics and Space Administration	Journals, conference papers, Proc., technical reports	Aeronautics, computer science, physics, space science
NEEDS (National Engineering Education Delivery System)	www.needs.org	Synthesis: A National Engineering Education Coalition	Lectures, articles, multimedia materials	Engineering, chemistry, computer science
WCR (Web Characterization Repository)	repository.cs.vt.edu	W3C Web Characterization Activity	J.s, conference papers, Proc., technical reports, books	Computer science - WWW

APPENDIX B

DTD FOR DLDL XML SPECIFICATION

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE DLDL [
<!ELEMENT DLDL (TITLE,DOCID,BASEURL,DLIBINFO,SEARCHDATA)>
<!ATTLIST DLDL   VersionNum CDATA #REQUIRED>
<!ELEMENT TITLE (#PCDATA)>
<!ATTLIST TITLE   Title CDATA #REQUIRED>
<!ELEMENT DOCID (REFNUM,REFDATE)>
<!ELEMENT REFNUM (#PCDATA)>
<!ATTLIST REFNUM   Title CDATA #REQUIRED>
<!ELEMENT REFDATE (#PCDATA)>
<!ATTLIST REFDATE   Title CDATA #FIXED "Document Date:">
<!ELEMENT BASEURL (#PCDATA)>
<!ATTLIST BASEURL   Title CDATA "Base URL:">
<!ELEMENT DLIBINFO (ORGANISATION,ARCHIVAL-TYPE*,SUBJECT*)>
<!ELEMENT ORGANISATION (#PCDATA)>
<!ATTLIST ORGANISATION   Title CDATA "Organisation:">
<!ELEMENT ARCHIVAL-TYPE (#PCDATA)>
<!ATTLIST ARCHIVAL-TYPE   Title CDATA "Archival Type:">
<!ELEMENT SUBJECT (#PCDATA)>
<!ATTLIST SUBJECT   Title CDATA "Subject:">
<!ELEMENT SEARCHDATA (REPLACE-FIELD,SEARCH-METHOD,SEARCH-
URL,FORMFIELD*,OUTPUTDATA,DOCHIT,MULTIPAGE)>
<!ATTLIST SEARCHDATA   Title CDATA #REQUIRED>
<!ELEMENT REPLACE-FIELD (#PCDATA)>
<!ATTLIST REPLACE-FIELD   Title CDATA "Number of fields to replace:">
<!ELEMENT SEARCH-METHOD (#PCDATA)>
<!ATTLIST SEARCH-METHOD   Title CDATA "Search Method:">
<!ELEMENT SEARCH-URL (#PCDATA)>
<!ATTLIST SEARCH-URL   Title CDATA "Search URL:">
<!ELEMENT FORMFIELD
(REQUIRED,WEIGHT,TYPE,LABEL,LENGTH,INPUTNAME,INPUTTYPE,INPUTVALUE)>
<!ELEMENT REQUIRED (#PCDATA)>
<!ATTLIST REQUIRED   Title CDATA "Required Field or not:">
<!ELEMENT WEIGHT (#PCDATA)>
<!ATTLIST WEIGHT   Title CDATA "Weight of Field:">
<!ELEMENT TYPE (#PCDATA)>
<!ATTLIST TYPE   Title CDATA "Search Criteria or Display Option:">
<!ELEMENT LABEL (#PCDATA)>
<!ATTLIST LABEL   Title CDATA "Displayed Field Name:">
<!ELEMENT LENGTH (#PCDATA)>
<!ATTLIST LENGTH   Title CDATA "Field Length:">
<!ELEMENT INPUTNAME (INPUTNAME_VALUE, INPUTNAME_MAPPING)>
<!ELEMENT INPUTNAME_VALUE (#PCDATA)>
<!ATTLIST INPUTNAME_VALUE   Title CDATA "internal Form Name:">
<!ELEMENT INPUTNAME_MAPPING (#PCDATA)>
<!ATTLIST INPUTNAME_MAPPING   Title CDATA "Mapped UI Field Name:">
<!ELEMENT INPUTTYPE (#PCDATA)>
<!ATTLIST INPUTTYPE   Title CDATA "Form Type:">
<!ELEMENT INPUTVALUE (DEFAULTVALUE*,OPTIONALVALUE*)>
<!ELEMENT DEFAULTVALUE (DEFAULTVALUE_DISPLAY, DEFAULTVALUE_INTERNAL, MAPPING?)>
<!ELEMENT DEFAULTVALUE_DISPLAY (#PCDATA)>
<!ATTLIST DEFAULTVALUE_DISPLAY   Title CDATA "Displayed Default Value">
<!ELEMENT DEFAULTVALUE_INTERNAL (#PCDATA)>
<!ATTLIST DEFAULTVALUE_INTERNAL   Title CDATA "internal Default Value">
<!ELEMENT MAPPING (#PCDATA)>
<!ATTLIST MAPPING   Title CDATA "internal Value MAPPING">

```

APPENDIX B (continued)

```

<!ELEMENT OPTIONALVALUE (OPTIONALVALUE_DISPLAY, OPTIONALVALUE_INTERNAL,
MAPPING?)>
<!ELEMENT OPTIONALVALUE_DISPLAY (#PCDATA)>
<!ATTLIST OPTIONALVALUE_DISPLAY Title CDATA "Displayed Optional Value">
<!ELEMENT OPTIONALVALUE_INTERNAL (#PCDATA)>
<!ATTLIST OPTIONALVALUE_INTERNAL Title CDATA "internal Optional Value"
<!ELEMENT OUTPUTDATA (OVAR-TAG,OVAR-MATCH*,OVAR-EXCLUDE-MATCH*,COMMENT-
MATCH-START,COMMENT-MATCH-END,RESULT-METADATA*,RECORD-METADATA*)>
<!ATTLIST OUTPUTDATA Title CDATA #REQUIRED>
<!ELEMENT OVAR-TAG (#PCDATA)>
<!ATTLIST OVAR-TAG Title CDATA "Output Tag:">
<!ELEMENT OVAR-MATCH (#PCDATA)>
<!ATTLIST OVAR-MATCH Title CDATA "Output Match:">
<!ELEMENT OVAR-EXCLUDE-MATCH (#PCDATA)>
<!ATTLIST OVAR-EXCLUDE-MATCH Title CDATA "Output Excluded Match:">
<!ATTLIST OVAR-EXCLUDE-MATCH EXACTMATCH CDATA "Y or N">
<!ELEMENT COMMENT-MATCH-START (#PCDATA)>
<!ATTLIST COMMENT-MATCH-START Title CDATA "the begining of matching string of result comment">
<!ELEMENT COMMENT-MATCH-END (#PCDATA)>
<!ATTLIST COMMENT-MATCH-END Title CDATA "the end of matching string of result comment">
<!ELEMENT RESULT-METADATA (MATCH-START,MATCH-
END,EXCLUDE*,REPLACE*,DELIMITER*,METADATA-FIELD*)>
<!ATTLIST RESULT-METADATA Title CDATA #REQUIRED>
<!ATTLIST RESULT-METADATA hasRecordLevel (true | false) #REQUIRED>
<!ELEMENT RECORD-METADATA (MATCH-START?,MATCH-
END?,EXCLUDE*,REPLACE*,DELIMITER*,METADATA-FIELD*)>
<!ATTLIST RECORD-METADATA Title CDATA #REQUIRED>
<!ELEMENT MATCH-START (#PCDATA)>
<!ATTLIST MATCH-START Title CDATA "the beginning of matching string of result metadata">
<!ATTLIST MATCH-START enforced (true | false) #IMPLIED>
<!ATTLIST MATCH-START isLastindex (true | false) #IMPLIED>
<!ELEMENT MATCH-END (#PCDATA)>
<!ATTLIST MATCH-END Title CDATA "the end of matching string of result metadata">
<!ATTLIST MATCH-END enforced (true | false) #IMPLIED>
<!ATTLIST MATCH-END isLastindex (true | false) #IMPLIED>
<!ELEMENT EXCLUDE (#PCDATA)>
<!ATTLIST EXCLUDE Title CDATA "the string should be excluded or removed when parsing">
<!ELEMENT REPLACE (OLD-STRING, NEW-STRING)>
<!ATTLIST REPLACE Title CDATA "replace old string with new string">
<!ELEMENT OLD-STRING (#PCDATA)>
<!ATTLIST OLD-STRING Title CDATA "the old string to be replaced">
<!ELEMENT NEW-STRING (#PCDATA)>
<!ATTLIST NEW-STRING Title CDATA "replace with the new string">
<!ELEMENT DELIMITER (#PCDATA)>
<!ATTLIST DELIMITER Title CDATA "delimiters to seperate metadata fields">
<!ELEMENT METADATA-FIELD (#PCDATA)>
<!ATTLIST METADATA-FIELD Title CDATA "information about a particular metadata field">
<!ATTLIST METADATA-FIELD order CDATA #IMPLIED>
<!ATTLIST METADATA-FIELD multiple (true | false) #IMPLIED>
<!ATTLIST METADATA-FIELD delimiter CDATA #IMPLIED>
<!ATTLIST METADATA-FIELD format CDATA #IMPLIED>
<!ATTLIST METADATA-FIELD null_value_string CDATA #IMPLIED>
<!ELEMENT DOCHIT (MATCHSTRING*,BEFORESTRING,AFTERSTRING)>
<!ATTLIST DOCHIT Title CDATA #REQUIRED>
<!ELEMENT MATCHSTRING (#PCDATA)>
<!ATTLIST MATCHSTRING Title CDATA "Match string for num of doc hits:">
<!ELEMENT BEFORESTRING (#PCDATA)>
<!ATTLIST BEFORESTRING Title CDATA "string before num of doc hits:">

```

APPENDIX B (continued)

```
<!ELEMENT AFTERSTRING (#PCDATA)>
<!ATTLIST AFTERSTRING  Title CDATA "string after num of doc hits:">
<!ELEMENT MULTIPAGE (MULTI-PAGE,HAS-NEXT,NEXT-URL,LINK-URL,URL-ADDITIONAL-
MATCH*,PAGE-HIT)>
<!ATTLIST MULTIPAGE  Title CDATA #REQUIRED>
<!ELEMENT MULTI-PAGE (#PCDATA)>
<!ATTLIST MULTI-PAGE  Title CDATA #REQUIRED>
<!ELEMENT HAS-NEXT (#PCDATA)>
<!ATTLIST HAS-NEXT  Title CDATA #REQUIRED>
<!ELEMENT NEXT-URL (#PCDATA)>
<!ATTLIST NEXT-URL  Title CDATA #REQUIRED>
<!ELEMENT LINK-URL (#PCDATA)>
<!ATTLIST LINK-URL  Title CDATA #REQUIRED>
<!ELEMENT URL-ADDITIONAL-MATCH (#PCDATA)>
<!ATTLIST URL-ADDITIONAL-MATCH  Title CDATA "Additional matching string for url matching">
<!ELEMENT PAGE-HIT (#PCDATA)>
<!ATTLIST PAGE-HIT  Title CDATA #REQUIRED>
]>
```


APPENDIX C

SAMPLE DDL SPECIFICATION FOR ACM

```

<DDL VersionNum="0003">
  <TITLE Title="Title:">Search on the ACM Digital Library</TITLE>
  <DOCID>
    <REFNUM Title="Document Reference Number:">DRNMXMLSPEC1.0ACM</REFNUM>
    <REFDATE Title="Document Date:">061902</REFDATE>
  </DOCID>
  <BASEURL Title="Base URL:">[Online]. Available: http://portal.acm.org</BASEURL>
  <DLIBINFO>
    <ORGANISATION Title="Organisation:">ACM Library</ORGANISATION>
    <ARCHIVAL-TYPE Title="Archival Type:">Select All: </ARCHIVAL-TYPE>
    <SUBJECT Title="Subject:">Select All:ALL</SUBJECT>
    <ADDITIONAL name="usecookie">true</ADDITIONAL>
  </DLIBINFO>
  <SEARCHDATA Title="Search info:">
    <REPLACE-FIELD Title="Number of fields to replace:">1</REPLACE-FIELD>
    <SEARCH-METHOD Title="Search Method:">POST</SEARCH-METHOD>
    <SEARCH-URL Title="Search URL:">[Online]. Available: http://portal.acm.org/results.cfm</SEARCH-URL>
    <FORMFIELD>
      <REQUIRED Title="Required Field or not:">Y</REQUIRED>
      <WEIGHT Title="Weight of Field:">1</WEIGHT>
      <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
      <LABEL Title="Displayed Field Name:">Search DL</LABEL>
      <LENGTH Title="Field Length:">35</LENGTH>
      <INPUTNAME>
        <INPUTNAME_VALUE Title="internal Form Name:">query</INPUTNAME_VALUE>
        <INPUTNAME_MAPPING Title="Mapped UI Field Name:">UI_keyword</INPUTNAME_MAPPING>
      </INPUTNAME>
      <INPUTTYPE Title="Form Type:">text input</INPUTTYPE>
      <INPUTVALUE/>
    </FORMFIELD>
    <FORMFIELD>
      <REQUIRED Title="Required Field or not:">Y</REQUIRED>
      <WEIGHT Title="Weight of Field:">1</WEIGHT>
      <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
      <LABEL Title="Displayed Field Name:">Collection</LABEL>
      <LENGTH Title="Field Length:">NULL</LENGTH>
      <INPUTNAME>
        <INPUTNAME_VALUE Title="internal Form Name:">coll</INPUTNAME_VALUE>
        <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
      </INPUTNAME>
      <INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
      <INPUTVALUE>
        <DEFAULTVALUE>
          <DEFAULTVALUE_DISPLAY Title="Displayed Default Value:">NULL</DEFAULTVALUE_DISPLAY>
          <DEFAULTVALUE_INTERNAL Title="internal Default Value:">ACM</DEFAULTVALUE_INTERNAL>
        </DEFAULTVALUE>
      </INPUTVALUE>
    </FORMFIELD>
    <FORMFIELD>
      <REQUIRED Title="Required Field or not:">Y</REQUIRED>
      <WEIGHT Title="Weight of Field:">1</WEIGHT>
      <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
      <LABEL Title="Displayed Field Name:">dl</LABEL>
      <LENGTH Title="Field Length:">NULL</LENGTH>
      <INPUTNAME>

```

APPENDIX C (continued)

```

<INPUTNAME_VALUE Title="internal Form Name:">dl</INPUTNAME_VALUE -->
<INPUTNAME_VALUE Title="internal Form Name:">whichdl</INPUTNAME_VALUE>
<INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
</INPUTNAME>
<INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
<INPUTVALUE>
  <DEFAULTVALUE>
    <DEFAULTVALUE_DISPLAY Title="Displayed Default
Value:">NULL</DEFAULTVALUE_DISPLAY>
    <DEFAULTVALUE_INTERNAL Title="internal Default
Value:">ACM</DEFAULTVALUE_INTERNAL>
  </DEFAULTVALUE>
</INPUTVALUE>
</FORMFIELD>
<OUTPUTDATA Title="ACM Output:">
  <OVAR-TAG Title="Output Tag:">A</OVAR-TAG>
  <OVAR-MATCH Title="Output Match:">citation.cfm</OVAR-MATCH>
  <OVAR-MATCH Title="Output Match:">class="medium-text"</OVAR-MATCH>
  <OVAR-EXCLUDE-MATCH Title="" EXACTMATCH="N">#FullText</OVAR-EXCLUDE-MATCH>
  <OVAR-EXCLUDE-MATCH Title="" EXACTMATCH="N">#CIT</OVAR-EXCLUDE-MATCH>
  <OVAR-EXCLUDE-MATCH Title="" EXACTMATCH="N">#references</OVAR-EXCLUDE-MATCH>
  <OVAR-EXCLUDE-MATCH Title="" EXACTMATCH="N">#abstract</OVAR-EXCLUDE-MATCH>
  <OVAR-EXCLUDE-MATCH Title="" EXACTMATCH="N">#indexterms</OVAR-EXCLUDE-
MATCH>
  <OVAR-EXCLUDE-MATCH Title="" EXACTMATCH="N">#citings</OVAR-EXCLUDE-MATCH>
  <OVAR-EXCLUDE-MATCH Title="" EXACTMATCH="N">#review</OVAR-EXCLUDE-MATCH>
  <COMMENT-MATCH-START Title="Comment match start:">A</COMMENT-MATCH-START>
  <COMMENT-MATCH-END Title="Comment match end:">relevancy" border="0"&gt;</COMMENT-
MATCH-END>
  <RESULT-METADATA Title="Result page metadata parsing:" hasRecordLevel="false">
    <MATCH-START enforced="true"&gt;&lt;div class="authors"&gt;</MATCH-START>
    <MATCH-END enforced="true"&gt;&lt;/div&gt;</MATCH-END>
    <EXCLUDE>&#13;</EXCLUDE>
    <EXCLUDE>&#10;</EXCLUDE>
    <EXCLUDE>&#9;</EXCLUDE>
    <METADATA-FIELD order="1" multiple="true" delimiter=",">CREATOR</METADATA-FIELD>
  </RESULT-METADATA>
  <RESULT-METADATA Title="Result page metadata parsing:" hasRecordLevel="false"><!-- 082003 -->
    <MATCH-START enforced="true"&gt;&lt;td class="small-text" nowrap&gt;</MATCH-START>
    <MATCH-END enforced="true"&gt;&lt;/td&gt;</MATCH-END>
    <EXCLUDE>&#13;</EXCLUDE>
    <EXCLUDE>&#10;</EXCLUDE>
    <EXCLUDE>&#9;</EXCLUDE>
    <EXCLUDE>&lt;br&gt;(&#10;)+?</EXCLUDE>
    <EXCLUDE>&lt;strong&gt;</EXCLUDE>
    <METADATA-FIELD order="1">DATE</METADATA-FIELD>
  </RESULT-METADATA>
  <RESULT-METADATA Title="Result page metadata parsing:" hasRecordLevel="false"><!-- 082003 -->
    <MATCH-START enforced="true"&gt;&lt;div class="addinfo"&gt;</MATCH-START>
    <MATCH-END enforced="true"&gt;&lt;/div&gt;</MATCH-END>
    <REPLACE>
      <OLD-STRING>&#10;</OLD-STRING>
      <NEW-STRING> </NEW-STRING>
    </REPLACE>
    <REPLACE>
      <OLD-STRING>&#13;</OLD-STRING>
      <NEW-STRING> </NEW-STRING>
    </REPLACE><!-- 082003

```

APPENDIX C (continued)

```

<EXCLUDE>#13;</EXCLUDE>
<EXCLUDE>#10;</EXCLUDE>-->
<EXCLUDE>#9;</EXCLUDE>
<EXCLUDE>&lt;br&gt;(.\\n)+?</EXCLUDE>
<EXCLUDE>&lt;strong&gt;</EXCLUDE>
<EXCLUDE>&lt;(.\\n)+?&gt;</EXCLUDE>
  <METADATA-FIELD order="1">PUBLICATION</METADATA-FIELD>
</RESULT-METADATA>
<RESULT-METADATA Title="Result page metadata parsing:" hasRecordLevel="false">
  <MATCH-START enforced="true">&lt;div class="abstract2"&gt;</MATCH-START>
  <MATCH-END enforced="true">&lt;br&gt;</MATCH-END>
  <EXCLUDE>#13;</EXCLUDE>
  <EXCLUDE>#10;</EXCLUDE>
  <EXCLUDE>#9;</EXCLUDE>
  <EXCLUDE>&lt;strong&gt;</EXCLUDE>
  <EXCLUDE>&lt;par&gt;</EXCLUDE>
  <METADATA-FIELD order="1">DESCRIPTION</METADATA-FIELD>
</RESULT-METADATA>
<RESULT-METADATA Title="Result page metadata parsing:" hasRecordLevel="false">
  <MATCH-START enforced="true">&lt;b&gt;Keywords&lt;/b&gt;</MATCH-START>
  <MATCH-END enforced="true">&lt;br&gt;</MATCH-END>
  <EXCLUDE>#13;</EXCLUDE>
  <EXCLUDE>#10;</EXCLUDE>
  <EXCLUDE>#9;</EXCLUDE>
  <EXCLUDE>&lt;strong&gt;</EXCLUDE>
  <EXCLUDE>&lt;par&gt;</EXCLUDE>
  <METADATA-FIELD order="1" multiple="true" delimiter=", ">KEYWORD</METADATA-FIELD>
</RESULT-METADATA>
</OUTPUTDATA>
<DOCHIT Title="Doc hits match string">
  <MATCHSTRING Title="Output Match:">Found</MATCHSTRING>
  <MATCHSTRING Title="Output Match:">searched.</MATCHSTRING>
  <BEFORESTRING Title="before string:">Found</BEFORESTRING>
  <AFTERSTRING Title="after string:">of</AFTERSTRING>
</DOCHIT>
<MULTIPAGE Title="Multi Page information">
  <MULTI-PAGE Title="MultiPage:">yes</MULTI-PAGE>
  <HAS-NEXT Title="Contains Next Link:">no</HAS-NEXT>
  <NEXT-URL Title="Matching String:">null</NEXT-URL>
  <LINK-URL Title="Matching String:">results.cfm?query=</LINK-URL>
  <PAGE-HIT Title="No. of hits per page:">20</PAGE-HIT>
</MULTIPAGE>
</SEARCHDATA>
</DLDL>

```

APPENDIX D

SAMPLE DDL SPECIFICATION FOR IEEE

```

<DDL VersionNum="0003">
  <TITLE Title="Title:">Search on the IEEE Digital Library</TITLE>
  <DOCID>
    <REFNUM Title="Document Reference Number:">DRNMXMLSPEC1.0IEEE</REFNUM>
    <REFDATE Title="Document Date:">101001</REFDATE>
  </DOCID>
  <BASEURL Title="Base URL:">[Online]. Available: http://www.ieee.org</BASEURL>
  <DLIBINFO>
    <ORGANISATION Title="Organisation:">IEEE Digital Library</ORGANISATION>
    <ARCHIVAL-TYPE Title="Archival Type:">Select All:null</ARCHIVAL-TYPE>
    <SUBJECT Title="Subject:">Select All:null</SUBJECT>
  </DLIBINFO>
  <SEARCHDATA Title="Search info:">
    <REPLACE-FIELD Title="Number of fields to replace:">2</REPLACE-FIELD>
    <SEARCH-METHOD Title="Search Method:">GET</SEARCH-METHOD>
    <SEARCH-URL Title="Search URL:">[Online]. Available:
http://odysseus.ieee.org/ieeesearch/query.html</SEARCH-URL>
    <FORMFIELD>
      <REQUIRED Title="Required Field or not:">Y</REQUIRED>
      <WEIGHT Title="Weight of Field:">1</WEIGHT>
      <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
      <LABEL Title="Displayed Field Name:">Request Type</LABEL>
      <LENGTH Title="Field Length:"/>
      <INPUTNAME>
        <INPUTNAME_VALUE Title="internal Form Name:">rq</INPUTNAME_VALUE>
        <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
      </INPUTNAME>
      <INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
      <INPUTVALUE>
        <DEFAULTVALUE>
          <DEFAULTVALUE_DISPLAY Title="Displayed Default
Value:">NULL</DEFAULTVALUE_DISPLAY>
          <DEFAULTVALUE_INTERNAL Title="internal Default Value:">0</DEFAULTVALUE_INTERNAL>
        </DEFAULTVALUE>
      </INPUTVALUE>
    </FORMFIELD>
    <FORMFIELD>
      <REQUIRED Title="Required Field or not:">Y</REQUIRED>
      <WEIGHT Title="Weight of Field:">1</WEIGHT>
      <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
      <LABEL Title="Displayed Field Name:">Collection</LABEL>
      <LENGTH Title="Field Length:">NULL</LENGTH>
      <INPUTNAME>
        <INPUTNAME_VALUE Title="internal Form Name:">col</INPUTNAME_VALUE>
        <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
      </INPUTNAME>
      <INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
      <INPUTVALUE>
        <DEFAULTVALUE>
          <DEFAULTVALUE_DISPLAY Title="Displayed Default
Value:">NULL</DEFAULTVALUE_DISPLAY>
          <DEFAULTVALUE_INTERNAL Title="internal Default
Value:">allieee</DEFAULTVALUE_INTERNAL>
        </DEFAULTVALUE>
      </INPUTVALUE>
    </FORMFIELD>
  </FORMFIELD>

```

APPENDIX D (continued)

```

<REQUIRED Title="Required Field or not:">Y</REQUIRED>
<WEIGHT Title="Weight of Field:">1</WEIGHT>
<TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
<LABEL Title="Displayed Field Name:">Keyword Name</LABEL>
<LENGTH Title="Field Length:">35</LENGTH>
<INPUTNAME>
  <INPUTNAME_VALUE Title="internal Form Name:">qt</INPUTNAME_VALUE>
  <INPUTNAME_MAPPING Title="Mapped UI Field Name:">UI_keyword</INPUTNAME_MAPPING>
</INPUTNAME>
<INPUTTYPE Title="Form Type:">text input</INPUTTYPE>
<INPUTVALUE/>
</FORMFIELD>
<FORMFIELD>
<REQUIRED Title="">Y</REQUIRED>
<WEIGHT Title="">1</WEIGHT>
<TYPE Title="">Search Criteria</TYPE>
<LABEL Title=""/>
<LENGTH Title=""/>
<INPUTNAME>
  <INPUTNAME_VALUE Title="">qc</INPUTNAME_VALUE>
  <INPUTNAME_MAPPING Title=""/>
</INPUTNAME>
<INPUTTYPE Title="">hidden</INPUTTYPE>
<INPUTVALUE>
  <DEFAULTVALUE>
    <DEFAULTVALUE_DISPLAY Title="Displayed Default
Value:">NULL</DEFAULTVALUE_DISPLAY>
    <DEFAULTVALUE_INTERNAL Title="internal Default
Value:">alliecc</DEFAULTVALUE_INTERNAL>
  </DEFAULTVALUE>
</INPUTVALUE>
</FORMFIELD>
<FORMFIELD>
<REQUIRED Title="Required Field or not:">Y</REQUIRED>
<WEIGHT Title="Weight of Field:">0.8</WEIGHT>
<TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
<LABEL Title="Displayed Field Name:">Number of Hits</LABEL>
<LENGTH Title="Field Length:">35</LENGTH>
<INPUTNAME>
  <INPUTNAME_VALUE Title="internal Form Name:">nh</INPUTNAME_VALUE>
  <INPUTNAME_MAPPING Title="Mapped UI Field Name:">UI_hits</INPUTNAME_MAPPING>
</INPUTNAME>
<INPUTTYPE Title="Form Type:">text input</INPUTTYPE>
<INPUTVALUE>
  <DEFAULTVALUE>
    <DEFAULTVALUE_DISPLAY Title="Displayed Default Value:">25</DEFAULTVALUE_DISPLAY>
    <DEFAULTVALUE_INTERNAL Title="internal Default Value:">25</DEFAULTVALUE_INTERNAL>
  </DEFAULTVALUE>
</INPUTVALUE>
</FORMFIELD>
<FORMFIELD>
<REQUIRED Title="">Y</REQUIRED>
<WEIGHT Title="">1</WEIGHT>
<TYPE Title="">Search Criteria</TYPE>
<LABEL Title=""/>
<LENGTH Title=""/>
<INPUTNAME>
  <INPUTNAME_VALUE Title="">ws</INPUTNAME_VALUE>

```

APPENDIX D (continued)

```

    <INPUTNAME_MAPPING Title=""/>
  </INPUTNAME>
  <INPUTTYPE Title="">hidden</INPUTTYPE>
  <INPUTVALUE>
    <DEFAULTVALUE>
      <DEFAULTVALUE_DISPLAY Title="Displayed Default Value:"/>
      <DEFAULTVALUE_INTERNAL Title="internal Default Value:">0</DEFAULTVALUE_INTERNAL>
    </DEFAULTVALUE>
  </INPUTVALUE>
</FORMFIELD>
<FORMFIELD>
  <REQUIRED Title="Required Field or not:">Y</REQUIRED>
  <WEIGHT Title="Weight of Field:">1</WEIGHT>
  <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
  <LABEL Title="Displayed Field Name:"/>
  <LENGTH Title="Field Length:"/>
  <INPUTNAME>
    <INPUTNAME_VALUE Title="internal Form Name:">qm</INPUTNAME_VALUE>
    <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
  </INPUTNAME>
  <INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
  <INPUTVALUE>
    <DEFAULTVALUE>
      <DEFAULTVALUE_DISPLAY Title="Displayed Default
Value:">NULL</DEFAULTVALUE_DISPLAY>
      <DEFAULTVALUE_INTERNAL Title="internal Default Value:">0</DEFAULTVALUE_INTERNAL>
    </DEFAULTVALUE>
  </INPUTVALUE>
</FORMFIELD>
<FORMFIELD>
  <REQUIRED Title="Required Field or not:">Y</REQUIRED>
  <WEIGHT Title="Weight of Field:">1</WEIGHT>
  <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
  <LABEL Title="Displayed Field Name:"/>
  <LENGTH Title="Field Length:"/>
  <INPUTNAME>
    <INPUTNAME_VALUE Title="internal Form Name:">st</INPUTNAME_VALUE>
    <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
  </INPUTNAME>
  <INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
  <INPUTVALUE>
    <DEFAULTVALUE>
      <DEFAULTVALUE_DISPLAY Title="Displayed Default
Value:">NULL</DEFAULTVALUE_DISPLAY>
      <DEFAULTVALUE_INTERNAL Title="internal Default Value:">1</DEFAULTVALUE_INTERNAL>
    </DEFAULTVALUE>
  </INPUTVALUE>
</FORMFIELD>
<FORMFIELD>
  <REQUIRED Title="Required Field or not:">Y</REQUIRED>
  <WEIGHT Title="Weight of Field:">1</WEIGHT>
  <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
  <LABEL Title="Displayed Field Name:"/>
  <LENGTH Title="Field Length:"/>
  <INPUTNAME>
    <INPUTNAME_VALUE Title="internal Form Name:">lk</INPUTNAME_VALUE>
    <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
  </INPUTNAME>

```

APPENDIX D (continued)

```

<INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
<INPUTVALUE>
  <DEFAULTVALUE>
    <DEFAULTVALUE_DISPLAY Title="Displayed Default
Value:">NULL</DEFAULTVALUE_DISPLAY>
    <DEFAULTVALUE_INTERNAL Title="internal Default Value:">1</DEFAULTVALUE_INTERNAL>
  </DEFAULTVALUE>
</INPUTVALUE>
</FORMFIELD>
<FORMFIELD>
  <REQUIRED Title="Required Field or not:">Y</REQUIRED>
  <WEIGHT Title="Weight of Field:">1</WEIGHT>
  <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
  <LABEL Title="Displayed Field Name:"/>
  <LENGTH Title="Field Length:"/>
  <INPUTNAME>
    <INPUTNAME_VALUE Title="internal Form Name:">rf</INPUTNAME_VALUE>
    <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
  </INPUTNAME>
  <INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
  <INPUTVALUE>
    <DEFAULTVALUE>
      <DEFAULTVALUE_DISPLAY Title="Displayed Default
Value:">NULL</DEFAULTVALUE_DISPLAY>
      <DEFAULTVALUE_INTERNAL Title="internal Default Value:">0</DEFAULTVALUE_INTERNAL>
    </DEFAULTVALUE>
  </INPUTVALUE>
</FORMFIELD>
<FORMFIELD>
  <REQUIRED Title="Required Field or not:">Y</REQUIRED>
  <WEIGHT Title="Weight of Field:">1</WEIGHT>
  <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
  <LABEL Title="Displayed Field Name:"/>
  <LENGTH Title="Field Length:"/>
  <INPUTNAME>
    <INPUTNAME_VALUE Title="internal Form Name:">rq2</INPUTNAME_VALUE>
    <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
  </INPUTNAME>
  <INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
  <INPUTVALUE>
    <DEFAULTVALUE>
      <DEFAULTVALUE_DISPLAY Title="Displayed Default
Value:">NULL</DEFAULTVALUE_DISPLAY>
      <DEFAULTVALUE_INTERNAL Title="internal Default Value:">0</DEFAULTVALUE_INTERNAL>
    </DEFAULTVALUE>
  </INPUTVALUE>
</FORMFIELD>
<OUTPUTDATA Title="IEEE Output:">
  <OVAR-TAG Title="Output Tag:">A</OVAR-TAG>
  <OVAR-MATCH Title="Output Match:">[Online]. Available: http://www.computer.org</OVAR-MATCH>
  <OVAR-MATCH Title="">[Online]. Available: http://www.ieee.org/organizations/pubs</OVAR-MATCH>
  <OVAR-MATCH Title="">[Online]. Available: http://www.ieee.org/organizations/society</OVAR-MATCH>
  <OVAR-MATCH Title="">[Online]. Available: http://www.ieee.org/web</OVAR-MATCH>
  <OVAR-MATCH Title="">[Online]. Available: http://www.ewh.ieee.org</OVAR-MATCH>
  <OVAR-MATCH Title="">[Online]. Available: http://www.comsoc.org</OVAR-MATCH>
  <OVAR-MATCH Title="">[Online]. Available: http://standards.ieee.org</OVAR-MATCH>
  <OVAR-MATCH Title="">[Online]. Available: http://www.ieeeusa.org</OVAR-MATCH>
  <OVAR-MATCH Title="">[Online]. Available: http://grouper.ieee.org</OVAR-MATCH>

```

APPENDIX D (continued)

```

<OVAR-MATCH Title="">ieee.org/</OVAR-MATCH>
<OVAR-MATCH Title="">computer.org/</OVAR-MATCH>
<OVAR-MATCH Title="">comsoc.org/</OVAR-MATCH>
<OVAR-EXCLUDE-MATCH Title="" EXACTMATCH="N">odysseus.ieee.org</OVAR-EXCLUDE-
MATCH>
  <COMMENT-MATCH-START Title="Comment match start:">span class=description</COMMENT-MATCH-
START>
  <COMMENT-MATCH-END Title="Comment match end:">font size="-1" class=fs</COMMENT-MATCH-
END>
  <RESULT-METADATA Title="Result page metadata parsing:" hasRecordLevel="false">
    <MATCH-START></MATCH-START>
    <MATCH-END enforced="true">&lt;/span&gt;&lt;br&gt;</MATCH-END>
    <EXCLUDE>&lt;(.|\\n)+?&gt;</EXCLUDE>
    <METADATA-FIELD order="1" multiple="false">DESCRIPTION</METADATA-FIELD>
  </RESULT-METADATA>
  <RESULT-METADATA Title="Result page metadata parsing:" hasRecordLevel="false">
    <MATCH-START enforced="true">class=publisher&gt;&lt;i&gt;</MATCH-START>
    <MATCH-END enforced="true">&lt;/i&gt;</MATCH-END>
    <EXCLUDE>&lt;(.|\\n)+?&gt;</EXCLUDE>
    <METADATA-FIELD order="1" multiple="false">PUBLISHER</METADATA-FIELD>
  </RESULT-METADATA>
  <RESULT-METADATA Title="Result page metadata parsing:" hasRecordLevel="false">
    <MATCH-START enforced="true">span class=date&gt;</MATCH-START>
    <MATCH-END enforced="true">&lt;/span&gt;</MATCH-END>
    <REPLACE>
      <OLD-STRING>&#38;nbsp;</OLD-STRING>
      <NEW-STRING> </NEW-STRING>
    </REPLACE>
    <EXCLUDE>&lt;(.|\\n)+?&gt;</EXCLUDE>
    <METADATA-FIELD order="1" multiple="false">DATE</METADATA-FIELD>
  </RESULT-METADATA>
</OUTPUTDATA>
<DOCHIT Title="Doc hits match string">
  <MATCHSTRING Title="Output Match:">found</MATCHSTRING>
  <BEFORESTRING Title="before string:">null</BEFORESTRING>
  <AFTERSTRING Title="after string:">results</AFTERSTRING>
</DOCHIT>
<MULTIPAGE Title="Multi Page information">
  <MULTI-PAGE Title="MultiPage:">no</MULTI-PAGE>
  <HAS-NEXT Title="Contains Next Link:">null</HAS-NEXT>
  <NEXT-URL Title="Matching String:">null</NEXT-URL>
  <LINK-URL Title="Matching String:">null</LINK-URL>
  <PAGE-HIT Title="No. of hits per page:">null</PAGE-HIT>
</MULTIPAGE>
</SEARCHDATA>
</DLDL>

```


APPENDIX E

SAMPLE DDL SPECIFICATION FOR NEEDS

```

<DDL VersionNum="0003">
  <TITLE Title="Title:">Search on the NEEDS Digital Library - MultiKeyword</TITLE>
  <DOCID>
    <REFNUM Title="Document Reference Number:">DRNMXMLSPEC1.0NEEDS</REFNUM>
    <REFDATE Title="Document Date:">991111</REFDATE>
  </DOCID>
  <BASEURL Title="Base URL:">[Online]. Available: http://www.needs.org</BASEURL>
  <DLIBINFO>
    <ORGANISATION Title="Organisation:">NEEDS Digital Library</ORGANISATION>
    <ARCHIVAL-TYPE Title="Archival Type:">Select All:null</ARCHIVAL-TYPE>
    <SUBJECT Title="Subject:">Select All:null</SUBJECT>
    <ADDITIONAL name="usecookie">>true</ADDITIONAL>
  </DLIBINFO>
  <SEARCHDATA Title="Search info:">
    <REPLACE-FIELD Title="Number of fields to replace:">2</REPLACE-FIELD>
    <SEARCH-METHOD Title="Search Method:">POST</SEARCH-METHOD>
    <SEARCH-URL Title="Search URL:">[Online]. Available:
    http://www.needs.org/needs/public/search/search_results/index.jhtml?_DARGS=/needs/public/search/index_body.jhtm
    l</SEARCH-URL>
  <FORMFIELD>
    <REQUIRED Title="Required Field or not:">Y</REQUIRED>
    <WEIGHT Title="Weight of Field:">1</WEIGHT>
    <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
    <LABEL Title="Displayed Field Name:">NULL</LABEL>
    <LENGTH Title="Field Length:">NULL</LENGTH>
    <INPUTNAME>      <INPUTNAME_VALUE Title="internal Form
    Name:">/smete/forms/FindLearningObjects.operation</INPUTNAME_VALUE>
    <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
    </INPUTNAME>
    <INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
    <INPUTVALUE>
      <DEFAULTVALUE>
        <DEFAULTVALUE_DISPLAY Title="Displayed Default
        Value:">NULL</DEFAULTVALUE_DISPLAY>
        <DEFAULTVALUE_INTERNAL Title="internal Default
        Value:">search</DEFAULTVALUE_INTERNAL>
      </DEFAULTVALUE>
    </INPUTVALUE>
  </FORMFIELD>
  <FORMFIELD>
    <REQUIRED Title="Required Field or not:">Y</REQUIRED>
    <WEIGHT Title="Weight of Field:">1</WEIGHT>
    <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
    <LABEL Title="Displayed Field Name:">NULL</LABEL>
    <LENGTH Title="Field Length:">NULL</LENGTH>
    <INPUTNAME>
      <INPUTNAME_VALUE Title="internal Form
      Name:">_D:/smete/forms/FindLearningObjects.operation</INPUTNAME_VALUE>
      <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
      </INPUTNAME>
      <INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
      <INPUTVALUE>
        <DEFAULTVALUE>
          <DEFAULTVALUE_DISPLAY Title="Displayed Default
          Value:">NULL</DEFAULTVALUE_DISPLAY>
          <DEFAULTVALUE_INTERNAL Title="internal Default Value:"> </DEFAULTVALUE_INTERNAL>
        </DEFAULTVALUE>
      </INPUTVALUE>
    </FORMFIELD>
  </FORMFIELD>

```

APPENDIX E (continued)

```

</INPUTVALUE>
</FORMFIELD>
<FORMFIELD>
  <REQUIRED Title="Required Field or not:">Y</REQUIRED>
  <WEIGHT Title="Weight of Field:">1</WEIGHT>
  <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
  <LABEL Title="Displayed Field Name:">Keywords</LABEL>
  <LENGTH Title="Field Length:">35</LENGTH>
  <INPUTNAME>
    <INPUTNAME_VALUE Title="internal Form
Name:">/smete/forms/FindLearningObjects.keyword</INPUTNAME_VALUE>
    <INPUTNAME_MAPPING Title="Mapped UI Field Name:">UI_keyword</INPUTNAME_MAPPING>
  </INPUTNAME>
  <INPUTTYPE Title="Form Type:">text input</INPUTTYPE>
  <INPUTVALUE/>
</FORMFIELD>
<FORMFIELD>
  <REQUIRED Title="Required Field or not:">Y</REQUIRED>
  <WEIGHT Title="Weight of Field:">1</WEIGHT>
  <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
  <LABEL Title="Displayed Field Name:">NULL</LABEL>
  <LENGTH Title="Field Length:">NULL</LENGTH>
  <INPUTNAME>
    <INPUTNAME_VALUE Title="internal Form
Name:">_D:/smete/forms/FindLearningObjects.keyword</INPUTNAME_VALUE>
    <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
  </INPUTNAME>
  <INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
  <INPUTVALUE>
    <DEFAULTVALUE>
      <DEFAULTVALUE_DISPLAY Title="Displayed Default
Value:">NULL</DEFAULTVALUE_DISPLAY>
      <DEFAULTVALUE_INTERNAL Title="internal Default Value:"> </DEFAULTVALUE_INTERNAL>
    </DEFAULTVALUE>
  </INPUTVALUE>
</FORMFIELD>
<FORMFIELD>
  <REQUIRED Title="Required Field or not:">Y</REQUIRED>
  <WEIGHT Title="Weight of Field:">1</WEIGHT>
  <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
  <LABEL Title="Displayed Field Name:">Grade</LABEL>
  <LENGTH Title="Field Length:">35</LENGTH>
  <INPUTNAME>
    <INPUTNAME_VALUE Title="internal Form
Name:">/smete/forms/FindLearningObjects.LearningResourceType</INPUTNAME_VALUE>
    <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
  </INPUTNAME>
  <INPUTTYPE Title="Form Type:">text input</INPUTTYPE>
  <INPUTVALUE>
    <DEFAULTVALUE>
      <DEFAULTVALUE_DISPLAY Title="Displayed Default Value:">All</DEFAULTVALUE_DISPLAY>
      <DEFAULTVALUE_INTERNAL Title="internal Default Value:"></DEFAULTVALUE_INTERNAL>
    </DEFAULTVALUE>
  </INPUTVALUE>
</FORMFIELD>
<FORMFIELD>
  <REQUIRED Title="Required Field or not:">Y</REQUIRED>
  <WEIGHT Title="Weight of Field:">1</WEIGHT>

```

APPENDIX E (continued)

```

<TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
<LABEL Title="Displayed Field Name:">NULL</LABEL>
<LENGTH Title="Field Length:">NULL</LENGTH>
<INPUTNAME>
  <INPUTNAME_VALUE Title="internal Form
Name: ">_D:/smete/forms/FindLearningObjects.learningResourceType</INPUTNAME_VALUE>
  <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
</INPUTNAME>
<INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
<INPUTVALUE>
  <DEFAULTVALUE>
  <DEFAULTVALUE_DISPLAY Title="Displayed Default
Value: ">NULL</DEFAULTVALUE_DISPLAY>
  <DEFAULTVALUE_INTERNAL Title="internal Default Value:"> </DEFAULTVALUE_INTERNAL>
</DEFAULTVALUE>
</INPUTVALUE>
</FORMFIELD>
<FORMFIELD>
  <REQUIRED Title="Required Field or not:">Y</REQUIRED>
  <WEIGHT Title="Weight of Field:">1</WEIGHT>
  <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
  <LABEL Title="Displayed Field Name:">Grade</LABEL>
  <LENGTH Title="Field Length:">35</LENGTH>
  <INPUTNAME>
  <INPUTNAME_VALUE Title="internal Form
Name: ">/smete/forms/FindLearningObjects.grade</INPUTNAME_VALUE>
  <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
</INPUTNAME>
  <INPUTTYPE Title="Form Type:">text input</INPUTTYPE>
  <INPUTVALUE>
  <DEFAULTVALUE>
  <DEFAULTVALUE_DISPLAY Title="Displayed Default Value:">All</DEFAULTVALUE_DISPLAY>
  <DEFAULTVALUE_INTERNAL Title="internal Default Value:">0-
Any</DEFAULTVALUE_INTERNAL>
  </DEFAULTVALUE>
</INPUTVALUE>
</FORMFIELD>
<FORMFIELD>
  <REQUIRED Title="Required Field or not:">Y</REQUIRED>
  <WEIGHT Title="Weight of Field:">1</WEIGHT>
  <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
  <LABEL Title="Displayed Field Name:">NULL</LABEL>
  <LENGTH Title="Field Length:">NULL</LENGTH>
  <INPUTNAME>
  <INPUTNAME_VALUE Title="internal Form
Name: ">_D:/smete/forms/FindLearningObjects.grade</INPUTNAME_VALUE>
  <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
</INPUTNAME>
  <INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
  <INPUTVALUE>
  <DEFAULTVALUE>
  <DEFAULTVALUE_DISPLAY Title="Displayed Default
Value: ">NULL</DEFAULTVALUE_DISPLAY>
  <DEFAULTVALUE_INTERNAL Title="internal Default Value:"> </DEFAULTVALUE_INTERNAL>
  </DEFAULTVALUE>
</INPUTVALUE>
</FORMFIELD>
<FORMFIELD>

```

APPENDIX E (continued)

```

<REQUIRED Title="Required Field or not:">N</REQUIRED>
<WEIGHT Title="Weight of Field:">0.9</WEIGHT>
<TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
<LABEL Title="Displayed Field Name:">Title</LABEL>
<LENGTH Title="Field Length:">35</LENGTH>
<INPUTNAME>
  <INPUTNAME_VALUE Title="internal Form
Name:">/smete/forms/FindLearningObjects.title</INPUTNAME_VALUE>
  <INPUTNAME_MAPPING Title="Mapped UI Field Name:">UI_title</INPUTNAME_MAPPING>
</INPUTNAME>
<INPUTTYPE Title="Form Type:">text input</INPUTTYPE>
<INPUTVALUE/>
</FORMFIELD>
<FORMFIELD>
<REQUIRED Title="Required Field or not:">Y</REQUIRED>
<WEIGHT Title="Weight of Field:">1</WEIGHT>
<TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
<LABEL Title="Displayed Field Name:">NULL</LABEL>
<LENGTH Title="Field Length:">NULL</LENGTH>
<INPUTNAME>
  <INPUTNAME_VALUE Title="internal Form
Name:">_D:/smete/forms/FindLearningObjects.title</INPUTNAME_VALUE>
  <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
</INPUTNAME>
<INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
<INPUTVALUE>
  <DEFAULTVALUE>
    <DEFAULTVALUE_DISPLAY Title="Displayed Default
Value:">NULL</DEFAULTVALUE_DISPLAY>
    <DEFAULTVALUE_INTERNAL Title="internal Default Value:"> </DEFAULTVALUE_INTERNAL>
  </DEFAULTVALUE>
</INPUTVALUE>
</FORMFIELD>
<FORMFIELD>
<REQUIRED Title="Required Field or not:">Y</REQUIRED>
<WEIGHT Title="Weight of Field:">1</WEIGHT>
<TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
<LABEL Title="Displayed Field Name:">Author/Creator</LABEL>
<LENGTH Title="Field Length:">35</LENGTH>
<INPUTNAME>
  <INPUTNAME_VALUE Title="internal Form
Name:">/smete/forms/FindLearningObjects.author</INPUTNAME_VALUE>
  <INPUTNAME_MAPPING Title="Mapped UI Field Name:">UI_creator</INPUTNAME_MAPPING>
</INPUTNAME>
<INPUTTYPE Title="Form Type:">text input</INPUTTYPE>
<INPUTVALUE/>
</FORMFIELD>
<FORMFIELD>
<REQUIRED Title="Required Field or not:">Y</REQUIRED>
<WEIGHT Title="Weight of Field:">1</WEIGHT>
<TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
<LABEL Title="Displayed Field Name:">NULL</LABEL>
<LENGTH Title="Field Length:">NULL</LENGTH>
<INPUTNAME>
  <INPUTNAME_VALUE Title="internal Form
Name:">_D:/smete/forms/FindLearningObjects.author</INPUTNAME_VALUE>
  <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
</INPUTNAME>

```

APPENDIX E (continued)

```

<INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
<INPUTVALUE>
  <DEFAULTVALUE>
    <DEFAULTVALUE_DISPLAY Title="Displayed Default
Value:">NULL</DEFAULTVALUE_DISPLAY>
    <DEFAULTVALUE_INTERNAL Title="internal Default Value:"> </DEFAULTVALUE_INTERNAL>
  </DEFAULTVALUE>
</INPUTVALUE>
</FORMFIELD>
<FORMFIELD>
  <REQUIRED Title="Required Field or not:">Y</REQUIRED>
  <WEIGHT Title="Weight of Field:">1</WEIGHT>
  <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
  <LABEL Title="Displayed Field Name:">Publication Year after</LABEL>
  <LENGTH Title="Field Length:">35</LENGTH>
  <INPUTNAME>
    <INPUTNAME_VALUE Title="internal Form
Name:">/smete/forms/FindLearningObjects.afterYear</INPUTNAME_VALUE>
    <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
  </INPUTNAME>
  <INPUTTYPE Title="Form Type:">text input</INPUTTYPE>
  <INPUTVALUE/>
</FORMFIELD>
<FORMFIELD>
  <REQUIRED Title="Required Field or not:">Y</REQUIRED>
  <WEIGHT Title="Weight of Field:">1</WEIGHT>
  <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
  <LABEL Title="Displayed Field Name:">NULL</LABEL>
  <LENGTH Title="Field Length:">NULL</LENGTH>
  <INPUTNAME>
    <INPUTNAME_VALUE Title="internal Form
Name:">_D:/smete/forms/FindLearningObjects.afterYear</INPUTNAME_VALUE>
    <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
  </INPUTNAME>
  <INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
  <INPUTVALUE>
    <DEFAULTVALUE>
      <DEFAULTVALUE_DISPLAY Title="Displayed Default
Value:">NULL</DEFAULTVALUE_DISPLAY>
      <DEFAULTVALUE_INTERNAL Title="internal Default Value:"> </DEFAULTVALUE_INTERNAL>
    </DEFAULTVALUE>
  </INPUTVALUE>
</FORMFIELD>
<FORMFIELD>
  <REQUIRED Title="Required Field or not:">Y</REQUIRED>
  <WEIGHT Title="Weight of Field:">1</WEIGHT>
  <TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
  <LABEL Title="Displayed Field Name:">Publication Year before</LABEL>
  <LENGTH Title="Field Length:">35</LENGTH>
  <INPUTNAME>
    <INPUTNAME_VALUE Title="internal Form
Name:">/smete/forms/FindLearningObjects.beforeYear</INPUTNAME_VALUE>
    <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
  </INPUTNAME>
  <INPUTTYPE Title="Form Type:">text input</INPUTTYPE>
  <INPUTVALUE/>
</FORMFIELD> <FORMFIELD>
  <REQUIRED Title="Required Field or not:">Y</REQUIRED>

```

APPENDIX E (continued)

```

<WEIGHT Title="Weight of Field:">1</WEIGHT>
<TYPE Title="Search Criteria or Display Option:">Search Criteria</TYPE>
<LABEL Title="Displayed Field Name:">NULL</LABEL>
<LENGTH Title="Field Length:">NULL</LENGTH>
<INPUTNAME>
  <INPUTNAME_VALUE Title="internal Form
Name:">/smete/forms/FindLearningObjects.search</INPUTNAME_VALUE>
  <INPUTNAME_MAPPING Title="Mapped UI Field Name:">NULL</INPUTNAME_MAPPING>
</INPUTNAME>
<INPUTTYPE Title="Form Type:">hidden</INPUTTYPE>
<INPUTVALUE>
  <DEFAULTVALUE>
    <DEFAULTVALUE_DISPLAY Title="Displayed Default
Value:">NULL</DEFAULTVALUE_DISPLAY>
    <DEFAULTVALUE_INTERNAL Title="internal Default
Value:">Search</DEFAULTVALUE_INTERNAL>
  </DEFAULTVALUE>
</INPUTVALUE>
</FORMFIELD>
<OUTPUTDATA Title="NEEDS Output:">
  <OVAR-TAG Title="Output Tag:">A</OVAR-TAG>
  <OVAR-MATCH Title="Output
Match:">needs/public/search/search_results/learning_resource/summary</OVAR-MATCH>
  <COMMENT-MATCH-START Title="Comment match start:">, </COMMENT-MATCH-START>
  <COMMENT-MATCH-END Title="Comment match end:">/p></COMMENT-MATCH-END>
  <RESULT-METADATA Title="Result page metadata parsing:" hasRecordLevel="true">
    <MATCH-START></MATCH-START>
    <MATCH-END enforced="true">&lt;/a&gt;</MATCH-END>
    <EXCLUDE>&#13;</EXCLUDE>
    <EXCLUDE>&#10;</EXCLUDE>
    <EXCLUDE>&#9;</EXCLUDE>
    <METADATA-FIELD multiple="false" >DATE</METADATA-FIELD>
  </RESULT-METADATA>
  <RECORD-METADATA Title="Record page metadata parsing:">
    <MATCH-START enforced="true">Title:&lt;/td&gt;</MATCH-START>
    <MATCH-END enforced="true">&lt;/td&gt;</MATCH-END>
    <EXCLUDE>&#13;</EXCLUDE>
    <EXCLUDE>&#10;</EXCLUDE>
    <EXCLUDE>&#9;</EXCLUDE>
    <EXCLUDE>&lt;!--(.\\n)+?--&gt;</EXCLUDE>
    <EXCLUDE>&lt;!(.\\n)+?&gt;</EXCLUDE>
    <METADATA-FIELD multiple="false" >TITLE</METADATA-FIELD>
  </RECORD-METADATA>
  <RECORD-METADATA Title="Record page metadata parsing:">
    <MATCH-START enforced="true">Authors:&lt;/td&gt;</MATCH-START>
    <MATCH-END enforced="true">&lt;/td&gt;</MATCH-END>
    <EXCLUDE>&#13;</EXCLUDE>
    <EXCLUDE>&#10;</EXCLUDE>
    <EXCLUDE>&#9;</EXCLUDE>
    <EXCLUDE>&lt;!--(.\\n)+?--&gt;</EXCLUDE>
    <EXCLUDE>&lt;!(.\\n)+?&gt;</EXCLUDE>
    <METADATA-FIELD multiple="true" delimiter="," null_value_string="[None
Found]">CREATOR</METADATA-FIELD>
  </RECORD-METADATA>
  <RECORD-METADATA Title="Record page metadata parsing:">
    <MATCH-START enforced="true">Courseware Series:&lt;/td&gt;</MATCH-START>
    <MATCH-END enforced="true">&lt;/td&gt;</MATCH-END>
    <EXCLUDE>&#13;</EXCLUDE>

```

APPENDIX E (continued)

```

    <EXCLUDE>&lt;!-(.\n)+?--&gt;</EXCLUDE>
    <EXCLUDE>&lt;!(.\n)+?&gt;</EXCLUDE>
    <METADATA-FIELD multiple="false" null_value_string="[Not a part of any series]">COURSEWARE
SERIES</METADATA-FIELD>
</RECORD-METADATA>
<RECORD-METADATA Title="Record page metadata parsing:">
  <MATCH-START enforced="true">Summary:&lt;/td&gt;</MATCH-START>
  <MATCH-END enforced="true">&lt;/td&gt;</MATCH-END>
  <EXCLUDE>&lt;!-(.\n)+?--&gt;</EXCLUDE>
  <EXCLUDE>&lt;!(.\n)+?&gt;</EXCLUDE>
  <METADATA-FIELD multiple="false" null_value_string="">DESCRIPTION</METADATA-FIELD>
</RECORD-METADATA>
<RECORD-METADATA Title="Record page metadata parsing:">
  <MATCH-START enforced="true">Keywords:&lt;/td&gt;</MATCH-START>
  <MATCH-END enforced="true">&lt;/td&gt;</MATCH-END>
  <EXCLUDE>&lt;!-(.\n)+?--&gt;</EXCLUDE>
  <EXCLUDE>&lt;!(.\n)+?&gt;</EXCLUDE>
  <METADATA-FIELD multiple="true" delimiter="," null_value_string="[Not
Defined]">KEYWORD</METADATA-FIELD>
</RECORD-METADATA>
<RECORD-METADATA Title="Record page metadata parsing:">
  <MATCH-START enforced="true">Subject Headings:&lt;/td&gt;</MATCH-START>
  <MATCH-END enforced="true">&lt;/td&gt;</MATCH-END>
  <EXCLUDE>&#13;</EXCLUDE>
  <EXCLUDE>&#10;</EXCLUDE>
  <EXCLUDE>&#9;</EXCLUDE>
  <EXCLUDE>&lt;!-(.\n)+?--&gt;</EXCLUDE>
  <EXCLUDE>&lt;!(.\n)+?&gt;</EXCLUDE>
  <METADATA-FIELD multiple="false" null_value_string="No subjects entered">SUBJECT</METADATA-
FIELD>
</RECORD-METADATA>
<RECORD-METADATA Title="Record page metadata parsing:">
  <MATCH-START enforced="true">Publishers:&lt;/td&gt;</MATCH-START>
  <MATCH-END enforced="true">&lt;/td&gt;</MATCH-END>
  <EXCLUDE>&#13;</EXCLUDE>
  <EXCLUDE>&#10;</EXCLUDE>
  <EXCLUDE>&#9;</EXCLUDE>
  <EXCLUDE>&lt;!-(.\n)+?--&gt;</EXCLUDE>
  <EXCLUDE>&lt;!(.\n)+?&gt;</EXCLUDE>
  <METADATA-FIELD multiple="true" delimiter="," null_value_string="">PUBLISHER</METADATA-
FIELD>
</RECORD-METADATA>
</OUTPUTDATA>
<DOCHIT Title="Doc hits match string">
  <MATCHSTRING Title="Output Match:">total results</MATCHSTRING>
  <BEFORESTRING Title="before string:">of</BEFORESTRING>
  <AFTERSTRING Title="after string:">total results</AFTERSTRING>
</DOCHIT>
<MULTIPAGE Title="Multi Page information">
  <MULTI-PAGE Title="MultiPage:">yes</MULTI-PAGE>
  <HAS-NEXT Title="Contains Next Link:">no</HAS-NEXT>
  <NEXT-URL Title="Matching String:">null</NEXT-URL>
  <LINK-URL Title="Matching String:">/needs/public/search/search_results/index.jhtml?queryId=</LINK-URL>
  <URL-ADDITIONAL-MATCH>page=</URL-ADDITIONAL-MATCH>
  <PAGE-HIT Title="No. of hits per page:">10</PAGE-HIT>
</MULTIPAGE>
</SEARCHDATA>
</DLDL>

```

VITA

Rong Shi

Department of Computer Science

Old Dominion University

Norfolk, VA 23529

ADDRESS

5349 Weblin Farm Road

Virginia Beach, VA 23455

EDUCATION

B.S. Information Measurement Technology and Instruments, July 1992, Shanghai

Jiao Tong University

M.S. Computer Science, March 1997, Shanghai Jiao Tong University

M.S. Computer Science, December 1999, Old Dominion University

Ph.D. Computer Science, December 2004, Old Dominion University