Designing Protocols in Support of Digital Library Componentization

Hussein Suleman and Edward A. Fox

Department of Computer Science, Virginia Tech, Blacksburg, VA, USA
{hussein,fox}@vt.edu
http://www.dlib.vt.edu/

Abstract. Reusability always has been a controversial topic in Digital Library (DL) design. While componentization has gained momentum in software engineering in general, there has not been broad DL standardization in component interfaces. Recently, the Open Archives Initiative (OAI) has begun to address this by creating a standard protocol for accessing metadata archives. We propose that the philosophy and approach adopted by the OAI can be extended easily to support inter-component protocols. In particular, we propose building DLs by connecting small components that communicate through a family of lightweight protocols, using XML as the data interchange mechanism. In order to test the feasibility of this, a set of protocols was designed based on the work of the OAI. Components adhering to these protocols were implemented and integrated into production and research DLs. The performance of these components was analyzed from the perspective of execution speed, network traffic, and data consistency. On the whole, this work has shown promise in the approach of applying the fundamental concepts of the OAI protocol to the task of DL component design and implementation.

1 Background and Motivation

As computers across the globe become part of the ever-expanding Internet, the communities of users and providers of information both grow. The providers of information contribute to increasing the body of information available to users, while the users, knowing this information exists, desire focused and instantaneous access to relevant information. The need to carefully manage collections of information contributed to the emergence of digital libraries (DLs), while the need to merge together collections to serve the needs of users has prompted the development of interoperability standards.

Special attention recently has been focused on the latter issue of interoperability with the emergence of the Open Archives Initiative (OAI) and its Protocol for Metadata Harvesting (PMH) [24]. The former issue of designing digital libraries to manage information has not received as much attention from the perspective of standardization. We propose in this paper that the philosophy and basic technical approach of the OAI can be applied to the design and construction of standardized components within digital libraries. Examples of such components include search engines, browsing services, annotation tools, peer review systems, and

recommendation systems. When connected together in a loosely coupled network to store data and provide services, such a collection of components constitutes an Open Digital Library (ODL), with many advantages over conventional DL architectures, notably: simplicity, reusability, and flexibility [22]. An example of such an ODL architecture is illustrated in Figure 1.

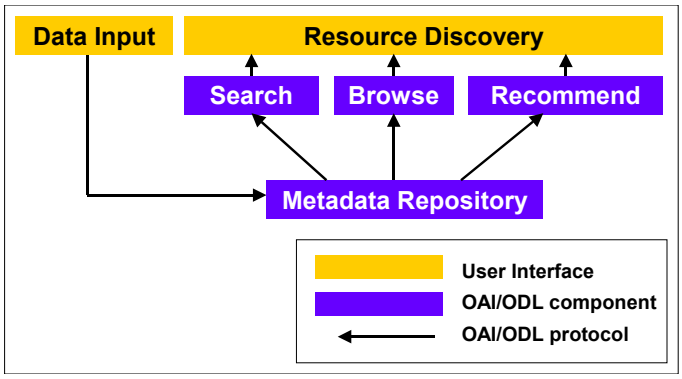

Fig. 1. Example ODL network of components

In keeping with current practices in software engineering, it long has been argued that DLs may benefit from software models based on object-oriented technology in general and componentization in particular [9]. Any such approach relies on an underlying component framework or set of application programming interfaces that are well defined and commonly known. Prior efforts have looked at various such mechanisms for inter-component communication.

Dienst [11] is a protocol (and software package) that used HTTP, and eventually XML, to provide for inter-component communication. Members of the Networked Computer Science Technical Reference Library [13] used earlier versions of Dienst for many years as the basis of their DL architecture and interoperability solution. (Dienst was one of the projects that served as a precursor to the current OAI-PMH.) The FEDORA project [17] further developed the Dienst repository architecture by defining abstract interfaces to structured digital objects, initially implemented over a CORBA communications medium.

The University of Michigan Digital Library Project [2] built DLs as collections of autonomous agents, with protocol-level negotiation to perform tasks collaboratively. The Stanford InfoBus project [1] wrapped its components into objects, with remote method invocation for communication.

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

2 Components and Requirements

One of our aims was to define a set of simple components that could be composed into production DL systems with minimal effort. To illustrate proof-of-concept, components were designed and developed to support the following common DL tasks:

- Submitting – adding an item to the system
- Searching – retrieving a list of items that correspond to a keyword query
- Browsing – retrieving a list of items that correspond to a set of categories
- Merging – combining multiple collections into one
- Annotating – adding comments and additional information to an item
- Recommending – retrieving a list of suggested items
- Rating – assigning a quantitative value to an item
- Reviewing – collaborative screening of items

In keeping with the OAI and ODL philosophies, these components were designed to be simple to deploy rather than complete according to a formal definition of their intended purposes. This approach also is taken by other interoperability protocols such as SDLIP [16], which wraps search systems with a common syntactic layer. However, SDLIP addresses only searching, which is just one aspect of the multitude of available services in modern DLs.

Each of the listed components needs to have a well-defined interface to communicate with other components. Table 1 lists gross requirements of some of the components in terms of their interfaces with other components.

Table 1. List of requirements for some ODL components

Component	Requirements
Search	Retrieve a list of items that match the supplied query. Add items to the search engine indices.
Browse	Retrieve a list of items that match a given set of criteria. Add items to the classification scheme.
Rating	Add a numerical rating for an item. Retrieve the numerical ratings, averages, and associated information for an item.
Annotate	Add an annotation to an item. Retrieve a list of annotations for an item.

The similarities in requirements suggest that a simpler model could be developed to factor out common features and incorporate those into a lower-level layer. The most basic operations needed for such a layer are the abilities to submit, retrieve, and delete items from a component or archive, as defined by Kahn and Wilensky [10] in their Repository Access Protocol. The ability to retrieve items is already provided by the OAI Protocol for Metadata Harvesting, so the approach taken was to first analyze this protocol and determine what needed to be modified or added in order to support the full range of functions needed for the identified DL components.

3 The OAI Protocol for Metadata Harvesting

The development of the OAI-PMH was a direct response to the need for simple interoperability standards [12], and this simplicity has led to adoption of the standard by many existing and new archives.

The OAI-PMH, commonly referred to as the OAI protocol, is a client-server protocol that is used to transfer XML-encoded records over an HTTP transport layer, with mechanisms to facilitate periodic updating. Table 2 lists the 6 service requests of this protocol that can be issued to obtain archive- or record-level metadata.

Table 2. OAI-PMH service requests and expected responses

Service Request	Expected Response
Identify	Description of archive: standards and protocols implemented
ListMetadataFormats	List of supported metadata formats
ListSets	List of archive sets and subsets
ListIdentifiers	List of record identifiers, optionally corresponding to a specified set and/or date range
GetRecord	Single metadata record corresponding to a specified identifier and in a specified metadata format
ListRecords	List of metadata records corresponding to a specified metadata format and, optionally, a set and/or date range

Archives that function as data providers implement the server end of this protocol and respond to these service requests, while those which wish to import or harvest data from data providers implement the client logic. These two pieces fit together to support simple metadata-transfer interoperability between archives.

4 Extensibility of the OAI Protocol

The OAI protocol is specifically aimed at the transfer of metadata among network-accessible devices. The mission of the OAI does not extend to supporting fine-grained inter-component interaction so the protocol was not designed with this in mind. However, since many of the requirements for such component protocols are already met by the OAI-PMH, it is possible and desirable to design new protocols based on the OAI-PMH, but with different purposes and somewhat different semantics. In keeping with this philosophy of reuse, we have looked into the development of new protocols as extensions of the OAI-PMH for inter-component communication.

The OAI-PMH already defines a specialized set of simple semantics for data access. Building on these semantics has the potential for greater impact on system

developers because the baseline OAI-PMH semantics are becoming increasingly well known in the DL community [14].

In order to design DL component interaction protocols based on the OAI-PMH it was first necessary to analyze the features that made this feasible or infeasible. Table 3 lists protocol features that were identified as supporting extension, those that need to be added to support extension, and those that inhibit extension. This list of features applies only to v1.1 of the OAI-PMH - later versions such as v2.0 address some of these. Further discussion of these features can be found in [23].

Table 3. Features of OAI protocol that affect extensibility

Supporting	Missing	Inhibiting
1 Set organization	5 Response-level containers	7 Harvesting granularity
2 GetRecord access	6 Submission	8 DC requirement
3 Metadata containers		
4 Identification containers		

Taking these into account, we propose a new protocol [23] to act as the underlying layer for component interaction protocols. This new protocol, the Extended OAI-PMH (XOAI-PMH), is an extension to v1.1 of the OAI-PMH, to exploit its inherent extensibility and attempt to overcome the stated limitations. XOAI-PMH involves four general syntax changes and one service request addition to OAI-PMH - a PutRecord analogue to the GetRecord request. XOAI-PMH is thus a different protocol from OAI-PMH, with a different purpose and different semantics. We do not propose XOAI-PMH as a replacement for OAI-PMH, but rather as an independent protocol for inter-component communication.

5 Open Digital Libraries

This new protocol fulfils the role of a baseline Repository Access Protocol, as defined by Kahn and Wilensky [10], in each component of the DL. More specific semantics then can be layered upon this to support the differing individual requirements of each component as discussed earlier. Ultimately, the components can be integrated into a configurable Open Digital Library of loosely connected and independent data and service providers, such as is shown in Figure 1.

Individual protocols were defined, as specialized versions of the XOAI-PMH, to meet the requirements of each component. Brief summaries of some of these specialized protocols follow.

5.1 The ODL-Search Protocol

Queries are encapsulated in *ListRecords* and *ListIdentifiers* service requests, with the list of keywords encoded into the set parameter along with the query language and bounds for the range of results to be returned. An example of such a query is:

```
…verb=ListIdentifiers&set=odlsearch1/computer
science/1/10
```

In order to acquire records to be indexed, the component may harvest records using OAI-PMH or XOAI-PMH.

5.2 The ODL-Browse Protocol

Just as in ODL-Search, *ListRecords* and *ListIdentifiers* are used to obtain lists of records, with the set parameter encoding the categories and sort order. An example of a query is:

```
…verb=ListIdentifiers&set=odlbrowse1/type(Computer)sort
(Year)/11/20
```

ListSets returns a list of all categories that may be used in browsing queries.
A Browse component also may harvest records using OAI-PMH or XOAI-PMH.

5.3 The ODL-Rate Protocol

PutRecord is used to add a rating for an item in the form of a metadata record encapsulating the numerical rating and the item identifier. An example of this record is:

```
<odl_rating>
   <subject>oai:People:a@b.com </subject>
   <object>oai:VTETD:12345</object>
   <rating>12</rating>
</odl_rating>
```

"subject" identifies the person submitting the rating while "object" identifies the item being rated.

Thereafter, *GetRecord* may be used to retrieve the individual ratings or an average value by specifying the "odl_rating_average" metadataPrefix for an item. An example of the metadata returned is:

```
<odl_rating_average>
   <average>12</average>
   <count>1</count>
</odl_rating_average>
```

When retrieving individual ratings, the set parameter is used to specify the item for which to return records.

5.4 The ODL-Annotate Protocol

PutRecord is used to add arbitrary annotations to the component, with the identifier of the item being annotated supplied as the set parameter (where that item could itself be a prior annotation). *ListRecords* and *ListIdentifiers* then list all annotations for an

item in reverse date order, with proper ordering maintained by the component – the set parameter is used to specify the item for which the "set of annotations" is requested as well as the range of entries to return. An example of such a request is:

…verb=ListIdentifiers&set=21/25/oai:VTETD:12345

In this case, the subset of annotations, starting at the 21st entry and ending at the 25th entry, is returned for the item identified by "oai:VTETD:12345".

Additional information about the item is provided using "about" containers for each record.

6 ODL Experimental Applications

In order to test the feasibility of the proposed componentized architecture for DLs using real world scenarios, a suite of components was implemented to support basic DL services.

6.1 Methodology

Components were implemented in accordance with the following protocols:

- ODL-Union, to merge together data from multiple OAI-compliant sources
- ODL-Filter, to filter OAI sources for illegal characters and non-unique identifiers
- ODL-Search, to index words in the metadata and permit search operations
- ODL-Browse, to sort and categorize data and permit browse operations
- ODL-Recent, to keep track of recently added items
- ODL-Annotate, to attach comments to an item
- ODL-Review, to keep track of peer-review workflow
- ODL-Submit, to accept submissions of items

The Union, Search, Browse, Filter, and Recent components were integrated into a simple user interface for the NDLTD system as shown in Figures 2 and 3, using metadata corresponding to Electronic Theses and Dissertations [21]. Figures 4 and 5 similarly display the user interface and architecture of the Browse component that was incorporated into the legacy DL system of the Computer Science Teaching Center (CSTC) [6].

Fig. 2. User interface of NDLTD ODL

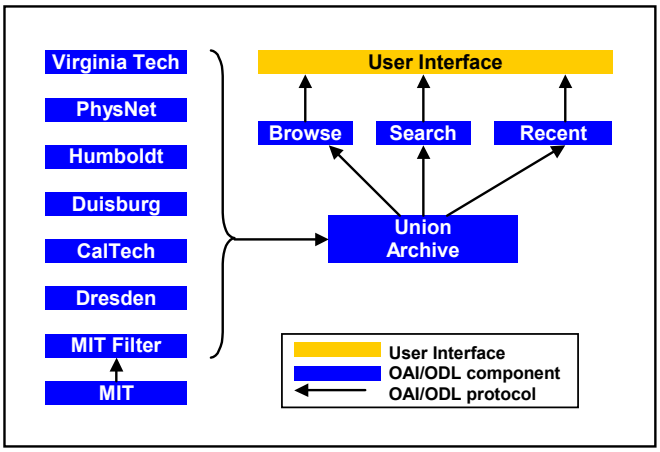

Fig. 3. Architecture of the NDLTD ODL

Fig. 4. User interface for the browsing function of CSTC

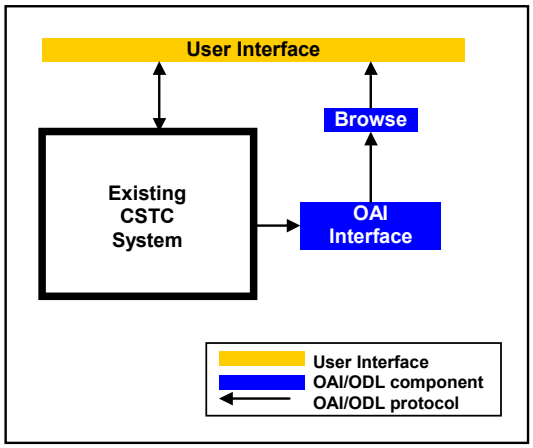

Fig. 5. Architecture of the CSTC system, incorporating Browse ODL component

For early testing, these prototype components were all derived from the original OAI protocol rather than the XOAI protocol, to allow for the use of existing testing and validation tools like the Repository Explorer [20]. Thus, using response-level containers, such information was embedded into other unused fields in the responses. Later tests involving the Annotate and Review components were fully conformant with the respective protocols. Figure 6 is an architectural overview of a general-purpose threaded discussion board based on the ODL-Annotate protocol. Figure 7 illustrates the back-end component architecture of a peer-review system originally

under development for the ACM Journal of Educational Resources in Computing. This component is generally useful for e-journal publishing and is currently being adapted for integration into the CSTC system.

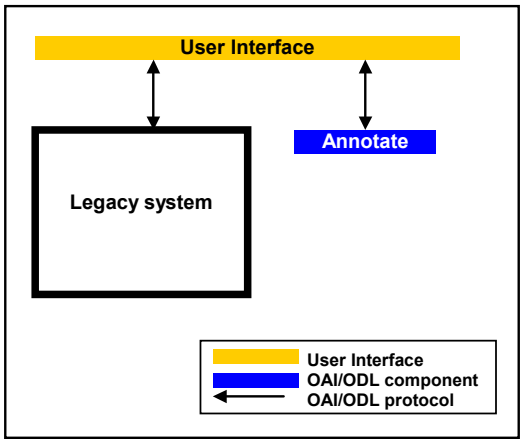

Fig. 6. ODL architecture of annotation system

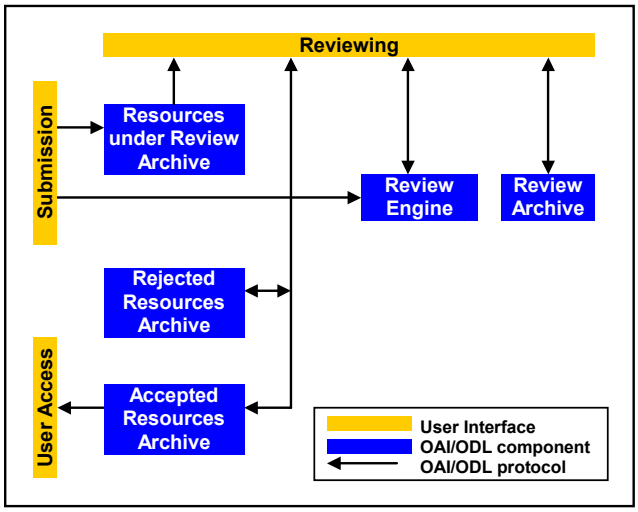

Fig. 7. ODL architecture of peer review system

6.2 Component Composition

The sequence of interactions corresponding to a typical use of a Search component is illustrated in Figure 8. The simplified ODL network consists of a source of data in

the form of an OAI-compliant archive and a Search component. The user interface layer is made up of a client's Web browser and the Web server, with scripts to generate HTML pages and forward requests to the ODL network.

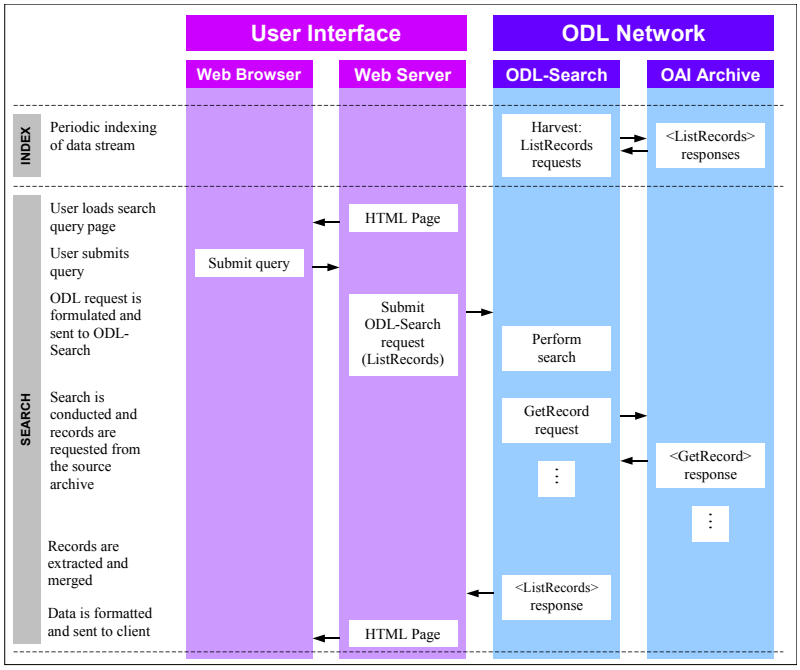

Fig. 8. Interface and component interaction during indexing and search operations

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

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

7 Harvesting and Propagation of Data

Interacting components inherit some of the performance characteristics of the OAI protocol, but also incur additional penalties that stem from the chaining together of components where each behaves asynchronously and, perhaps, remotely. We explored some of the concerns and related solutions.

For components that required an input stream of records from an OAI/XOAI archive or component, we chose the least network-intensive harvesting algorithm - using the *ListRecords* service request instead of the combination of *ListIdentifiers* and *GetRecord*. While the latter is arguably more robust, the former approach is faster, and within the context of a single system (located on a single machine or machines which are located in the same physical environment), speed is more important than network robustness.

Consistency of data also is an important issue and in this regard networked components suffer from the same problems as hierarchical Open Archives. By using a finer timestamp granularity in the baseline XOAI-PMH, we have decreased the effects of this problem. However, if metadata in one component changes and a downstream component does not synchronize immediately then there will be temporary inconsistencies. We are currently investigating ways of minimizing these inconsistencies using additional communication among components.

8 Component Speed Enhancements

While the aim of componentization is to make development simpler and repeatable, this cannot be at the expense of reduced functionality or efficiency. The time taken for inter-component communication can be reduced either by reducing the number of network requests or by changing the types of requests to maximize network utilization.

Various approaches were investigated to increase speed without sacrificing the advantages of a componentized system. The most successful and promising solutions found to maximize network utilization and minimize the processing delay normally associated with executing Web applications are discussed below.

8.1 Caching

Using caching at various levels within the experimental systems resulted in speed improvements. For example, the Browse component cached the results from the Union component, thus minimizing the number of recurring requests. Secondly, the user interface cached the responses to most requests; thus speeding up the process of browsing through a list of returned items. Together, these had a noticeable effect on system performance. One problem that manifested itself was that of stale data in a cache. It is still being investigated – there are ways to force a refresh from the Web browser to propagate to the server's scripts, but this apparently only works for Netscape browsers and works differently in each version.

8.2 FastCGI

FastCGI [4] is an add-on kit that provides persistent script capabilities to a Web server, independently of the programming language. Scripts need to be modified slightly by encapsulating them in a simple loop but this is relatively minor and for some components it was possible to create both regular and FastCGI versions without much change. FastCGI provides an add-on server module that loads a script on demand and keeps it persistent, with support for dynamic reloading and dynamic load balancing. This was tested for some components. There were additional security problems that needed to be resolved since FastCGI enforced a higher level of security than regular scripts, but better programming discipline and security is good for component development, so this can be seen as another advantage.

8.3 SpeedyCGI

Without modification to the Web server, it is possible for a component to stay resident in memory and be glued into the Web server whenever necessary by a much smaller program. This is the approach taken by the SpeedyCGI toolkit [19], which improves performance without any modification of the source code. Unlike the other approaches, this toolkit only worked with the Perl language, but the technique is generally applicable to any development environment.

8.4 Batch Requests

At a protocol level it is possible to reduce the number of requests by combining responses. In the ODL-Review protocol, the reviewable items visible to an editor are listed using *GetRecord* – the response is an XML container that contains within it many individual records. Similarly, the ODL-Union protocol may be extended in the future to support requesting multiple records with a single *ListRecords* request that specifies a combined list of identifiers encoded within a single set parameter.

9 Future Work and Conclusions

9.1 Development and Refinement of Component Libraries and Protocols

We have developed sample protocol designs for many existing digital library use cases, including searching, browsing, threaded discussions, peer review, recommendation, and rating systems. These and additional components will be integrated into existing and new DL systems to test for reusability and portability.

This set of designs will be re-evaluated in light of recent developments in the OAI-PMH. Some of the issues that currently need to be addressed by the XOAI protocol may be irrelevant if they are incorporated into a future OAI protocol, as we have suggested by way of our involvement in the OAI technical and steering committees.

In addition, we will attempt to integrate our work with emerging standards in web-based services such as SOAP [3] and WSDL [15], which are expected to provide a general syntactic layer for high-level application protocols.

Our prototyping work has demonstrated some feasible component designs. These will be extended to other components, with additional generality introduced wherever possible. Further work will be done on separating instances of components from configuration information – ultimately allowing for the possibility of a suite of components servicing multiple DLs, and visual composition of components.

The VIDI protocol [25] for connecting visualization components to digital libraries is being independently developed to co-exist with and build upon ODL components.

9.2 Component Testing and Validation

Testing of the OAI protocol is largely supported by the Repository Explorer [20], which we developed specifically for the purposes of validation of requests and responses and standardization of implementations.

This software will be extended to support the additional functionality of the ODL protocols by building in support for the XOAI protocol. This tool would then support the development of components using the ODL protocols. Particular support for individual ODL protocols is also an option if the software can be specialized to test for more specific semantics based on specifications.

9.3 Evaluation

Further evaluation of the feasibility of building Digital Libraries as networks of extended Open Archives will be carried out in terms of their equivalence to monolithic systems, extensibility of components, and usability of the component model. Performance evaluation is an ongoing process, and further work is being done on:

- Communications and protocol overhead incurred by OAI/XOAI/ODL protocols.
- Stability of the communications protocols relative to the datestamp granularities – evaluation of the trade-off between duplication of records and the possibility of missing records.
- Speed of the ODL networks compared with monolithic systems.
- Storage required for components and the effects of data duplication.
- Consistency among various copies of data stored on different nodes.
- Harvesting algorithms and their efficiencies in terms of speed and network utilization.

9.4 Conclusions

It is hoped that the ongoing results of this work will change the way people build digital libraries, so they can utilize simple and reusable component models based on already established standards. In particular, we hope our work will help lead to

"ODL-in-a-box" solutions that can be tailored to classes of applications, such as the National STEM Digital Library (www.nsdl.nsf.gov). Unlike other "DL-in-a-box" solutions like Eprints [8] and Repository-in-a-Box [18], ODL-based systems will be trivially extensible.

Building upon this foundation of extensibility, it then will be possible to work on providing more interesting services to users, thus bridging the wide gap between current research and production systems, and ultimately making information more accessible to people.

10 Acknowledgements

Thanks are given for the support of NSF through its grants: IIS-9986089, IIS-0002935, IIS-0080748, IIS-0086227, DUE-0121679, DUE-0121741, and DUE-0136690.

References

1. Baldonado, M., Chang, C. K., Gravano, L., and Paepcke, A. The Stanford Digital Library Metadata Architecture, in International Journal on Digital Libraries 1, 2 (1997), 108-121. Available http://www-diglib.stanford.edu/cgi-bin/get/SIDL-WP-1996-0051.
2. Birmingham, W. P. An Agent-Based Architecture for Digital Libraries, in D-Lib Magazine 1, 1 (July 1995). Available http://www.dlib.org/dlib/July95/07birmingham.html.
3. Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D. Simple Object Access Protocol (SOAP) v1.1, W3C Technical Note, (8 May 2000). Available http://www.w3.org/TR/SOAP/
4. Brown, M. R. FastCGI – A High-Performance Gateway Interface, position paper at "Programming the Web - a search for APIs" workshop, Fifth International World Wide Web Conference, (Paris, France, 6 May 1996). Available http://www.fastcgi.com/devkit/doc/www5-api-workshop.html
5. Clark, J. (editor) XSL Transformations Version 1.0, W3C Recommendation, (November 1999). Available http://www.w3.org/TR/xslt
6. Computer Science Teaching Center; www.cstc.org/. Accessed 26 June 2002
7. Dublin Core Metadata Initiative. Dublin Core Metadata Element Set Version 1.1: Reference Description, 1997. Available http://www.dublincore.org/documents/dces/.
8. EPrints; http://www.eprints.org/. Accessed 26 June 2002
9. Gladney, H., Ahmed, Z., Ashany, R., Belkin, N. J., Fox, E. A., and Zemankova, M. Digital Library: Gross Structure and Requirements (Report from a Workshop), IBM Almaden Research Center, Research Report RJ9840, May 1994. Available http://www.ifla.org.sg/documents/libraries/net/rj9840.pdf
10. Kahn, R., and Wilensky, R. A Framework for Distributed Digital Object Services, CNRI, 1995. Available http://www.cnri.reston.va.us/k-w.html.
11. Lagoze., C., and Davis, J. R. Dienst – An Architecture for Distributed Document Libraries, in Commun. ACM 38, 4 (April 1995), 47.
12. Lagoze, C., and Van de Sompel, H. The Open Archives Initiative: Building a low-barrier interoperability framework, in Proceedings of JCDL 2001 (Roanoke VA, June 2001), ACM Press, 54-62.

13. Leiner, B. M. The NCSTRL Approach to Open Architecture, in D-Lib Magazine 4, 11 (December 1998). Available http://www.dlib.org/dlib/december98/leiner/12leiner.html
14. Nichols, Bill. Open Meta Tools, in BYTE Magazine, 25 February 2002. Available http://www.byte.com/documents/s=7023/byt1014229948533/0225_nicholls.html
15. Ogbuji, U. Using WSDL in SOAP Applications, IBM developerWorks, (November 2000). Available http://www-106.ibm.com/developerworks/webservices/library/ws-soap/index.html
16. Paepcke, A., Brandriff, R., Janee, G., Larson, R., Ludaescher, B., Melnik, S., and Raghavan S. Search Middleware and the Simple Digital Library Interoperability Protocol, in D-Lib Magazine 6, 3 (March 2000). Available http://www.dlib.org/dlib/march00/paepcke/03paepcke.html
17. Payette, S., and Lagoze, C. Flexible and Extensible Digital Object and Repository Architecture, in Proceedings of Second European Conference on Research and Advanced Technology for Digital Libraries (Heraklion, Crete, Greece, September 21-23 1998), Springer, 1998, (Lecture notes in computer science; Vol. 1513).
18. Repository-in-a-Box; http://www.nhse.org/RIB/. Accessed 26 June 2002
19. SpeedyCGI; http://daemoninc.com/speedycgi/. Accessed 26 June 2002
20. Suleman, H. Enforcing Interoperability with the Open Archives Initiative Repository Explorer, in Proceedings of JCDL 2001, (Roanoke, VA, June 2001), ACM Press, 63-64.
21. Suleman, H., Atkins, A., Gonçalves, M. A., France, R. K., Fox, E. A., Chachra, V., Crowder, M., and Young, J. Networked Digital Library of Theses and Dissertations: Bridging the Gaps for Global Access - Part 1: Mission and Progress, and Part 2: Services and Research, in D-Lib Magazine 7, 9 (September 2001). Available http://www.dlib.org/dlib/september01/suleman/09suleman-pt1.html and http://www.dlib.org/dlib/september01/suleman/09suleman-pt2.html.
22. Suleman, H., and Fox, E. A. A Framework for Building Open Digital Libraries, in D-Lib Magazine 7, 12 (December 2001). Available http://www.dlib.org/dlib/december01/suleman/12suleman.html.
23. Suleman, H., and Fox, E. A. Beyond Harvesting: Digital Library Components as OAI Extensions, Technical Report, Department of Computer Science, Virginia Tech (January 2001).
24. Van de Sompel, H., and Lagoze, C. The Open Archives Initiative Protocol for Metadata Harvesting. Open Archives Initiative, 2001. Available http://www.openarchives.org/OAI_protocol/openarchivesprotocol.html.
25. Wang, Jun. VIDI: A Lightweight Protocol Between Visualization Tools and Digital Libraries, Master's Thesis, Virginia Tech (May 2002).