# Beyond Harvesting: Digital Library Components as OAI Extensions

**Hussein Suleman**
Department of Computer Science
Virginia Tech, Blacksburg, VA, USA
+1 540 231 3615
hussein@vt.edu

**Edward A. Fox**
Department of Computer Science
Virginia Tech, Blacksburg, VA, USA
+1 540 231 5113
fox@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 yet 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. It is proposed that this protocol be extended to act as the glue that binds together various components of a typical DL. In order to test the feasibility of this approach, a set of protocol extensions was created, implemented, and integrated as components of 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 simultaneously revealed the feasibility of such OAI extensions for component interaction, and has identified aspects of the OAI protocol that constrain such extensions.

**Keywords**
Interoperability, architecture, protocols, componentization

## BACKGROUND AND MOTIVATION

Digital libraries have challenged software engineers because of their loosely defined parameters and ever-changing requirements. One of these parameters has been the requirement for interoperability. The solution most recently adopted by many practitioners is the Protocol for Metadata Harvesting (PMH) [1], developed by the Open Archives Initiative (OAI). The development of this protocol was a direct response to the need for simple standards [2], 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, with mechanisms for periodic updating.

| 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 |

**Table 1. OAI-PMH service requests and expected responses**

Table 1 lists the 6 service requests of this protocol that can be issued to obtain archive or record-level metadata.

Archives that function as data providers implement the server end of this protocol as responses to these service requests, while those who 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.

This does not, however, ease the task of building and maintaining digital library (DL) software. 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 [3]. 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 [4] is a protocol (and software package) that uses HTTP and XML to provide for inter-component communication. Members of the Networked Computer Science Technical Reference Library [5] used this successfully for many years. (A lightweight version of Dienst was in fact the precursor to the current OAI protocol.) The complexity of Dienst as a complete package, however, led to an unwillingness of archivists to adopt the protocol and software. The FEDORA project [6] 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 [7] built DLs as collections of autonomous agents, with protocol-level negotiation to perform tasks collaboratively. The Stanford InfoBus project [8] 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 is a driving force behind interoperability protocols such as the OAI-PMH, which is used in this work as an alternative glue to bind together components in a larger DL.

Like any other communications mechanism, the OAI-PMH requires specialized semantics for particular components. These extensions, because of the OAI philosophy of simplicity, have the potential for greater impact on system developers because they build upon a baseline set of semantics that are becoming increasingly well known in the DL community. With these extensions, the OAI-PMH fulfils the role of a Repository Access Protocol, as defined by Kahn and Wilensky [9], in each component of the DL. Components then can be integrated into a configurable network of loosely connected and independent data and service providers, referred to as an Open Digital Library (ODL). Such a loose network of components, illustrated in Figure 1, has many advantages over tightly coupled systems – notably simplicity, reusability, and flexibility [10].

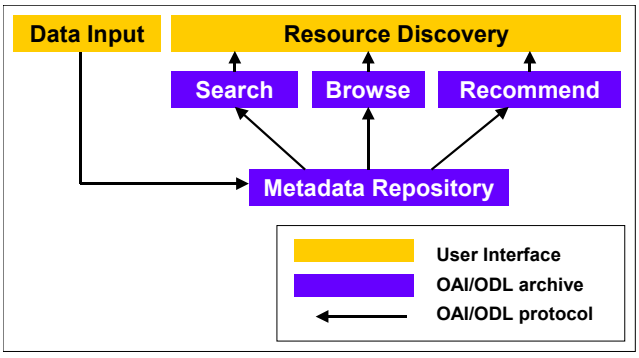


**Figure 1. Example ODL network of components**

This paper analyzes extensibility of the OAI protocol and then proposes a generalized extension and two specific protocols that support componentized DLs. Initial implementations and experiments are discussed and analyzed, leading to a natural course of ongoing work on this approach to DL architecture.

## EXTENSIBILITY OF THE OAI PROTOCOL

In order to create DL component interaction protocols based on the OAI protocol it is first necessary to analyze features that make this feasible or infeasible. Table 2 lists protocol features that were identified as supporting extension, those that need to be added to support extension, and those that inhibit extension.

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

**Table 2. Features of OAI protocol that affect extensibility**

### 1. Set Organization (Supporting)

OAI-PMH allows for archives to associate records with arbitrary sets. This can be exploited easily to select ranges or subsets of records from archives. If sets are generated dynamically, they also provide a mechanism to respond to arbitrary run-time queries.

### 2. GetRecord Repository Access (Supporting)

The GetRecord and ListMetadataFormats verbs allow access to one specific record in the archive. While this is not as useful as ListRecords for large-scale harvesting operations, it does provide a complete mechanism to retrieve records, in a random-access fashion.

### 3. Metadata Containers (Supporting)

Any format of metadata (that can be expressed in XML) can be transmitted using the OAI protocol. Thus, in addition to traditional metadata records, transactional data normally associated with the workflow of a DL can be stored in OAI-compliant archives if the data is cast into metadata records.

### 4. Identification Containers (Supporting)

The Identify verb supports the notion of containers, where additional information about an archive may be stored. These containers can be used to specify what extensions to the OAI protocol are supported.

**5. Response-Level Containers (Missing)**
Each OAI record has an XML container for embedding metadata about the record (such as rights information). However, no container exists to transmit information about the act of creating the response as a whole. This could be indispensable to return information such as result set cardinality or the total number of records in the response (when the response is broken up into chunks).

**6. Submission (Missing)**
The OAI protocol was designed to support exporting of a collection, but no mechanism was defined for data to be added. There are many issues related to security and quality control that are currently avoided because there is no way to add records to an archive. However, in the controlled environment of a single DL, these concerns are not as grave. With the introduction of a data input operation, archives could support local repository and database-like operations while still providing controlled global access at the level of the read-only OAI protocol.

**7. Harvesting Granularity (Inhibiting)**
The OAI protocol uses a datestamp in order to support harvesting by date, but the granularity is a single day since these datestamps contain only dates and not times. After taking into account the differences in timezones, a harvester has to overlap harvesting dates by at least one day in order not to miss any new entries. This is problematic for applications involving rapidly growing content or for situations in which there are frequent changes. One other solution is to use a finer granularity (e.g., time as well as date) and this is advocated as a simpler and more efficient solution.

**8. DC Requirement (Inhibiting)**
Every metadata record disseminated by the OAI protocol must have an equivalent Dublin Core Metadata Set [11] version. For gross-level interoperability among archives containing document-like objects this makes sense, but when the records stored in the archive are neither exposed to the outside world nor correspond to documents, it is irrelevant to map them to Dublin Core.

**THE EXTENDED OAI-PMH (XOAI-PMH)**
Many DL services can be encapsulated into Open Archives (archives supporting the OAI-PMH), with relevant extensions and specific semantics to support specialized functionality. Since some of this functionality is common to multiple components, these features are factored out into the general-purpose XOAI-PMH extension. This extension to the existing OAI-PMH exploits its inherent extensibility and attempts to overcome its limitations. XOAI-PMH involves four general changes and one service request addition to OAI-PMH.

**General Changes**

*DC Requirement*
Dublin Core is not a required metadata format.

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

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

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

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

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

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

**Service Request Addition**

*PutRecord (new optional service request)*
*Semantics:*

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

*Parameters:*

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

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

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

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

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

*Return Values:*

The response is a minimal OAI-PMH response containing just the requestURL and responseDate.

*Exceptions:*

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

*Example Request:*

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

*Example Response:*

```
<?xml version="1.0" ?>
<PutRecord>
<responseDate>2001-10-27T19:20:30-
05:00</responseDate>
<requestURL>http://an.oa.org/OAIscript?verb=Put
Record&metadataPrefix=oai dc&identifier=oai:ABC
:123&metadata=&lt;test%20xmlns%3D"testns"%20xsi
%3AschemaLocation%3D"testns%20testschema"&gt;&l
t;title&gt;aTitle&lt;%2Ftitle&gt;&lt;%2Ftest&gt
;</requestURL>
</PutRecord>
```

## THE ODL-UNION COMPONENT

### Description

In some instances metadata from multiple archives can be coalesced into a single archive to support specific requirements such as a reduction in network traffic where multiple services require the use of the metadata from remote locations. Also, where the archives are established to parallel an organization that is itself hierarchical, it may be useful to gather all the metadata into root nodes to provide centralized services across the entire organization (or subsets of it). Figure 2 illustrates a simple ODL network using the ODL-Union component.

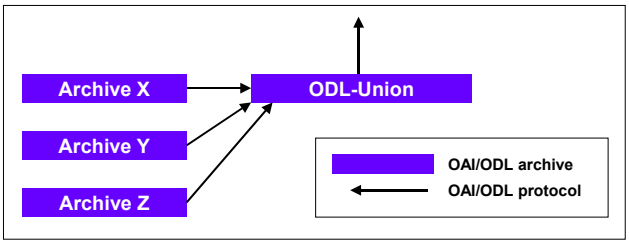


**Figure 2. Simple ODL network using ODL-Union**

The ODL-Union component harvests metadata from multiple sources and republishes it via a single XOAI-PMH interface. Parameters may be set to select the metadata formats and sets to harvest from each source archive. From the perspective of the ODL-Union component, the whole archive could be harvested, thus losing set information (unless extra processing is employed), or individual sets could be harvested with a more sophisticated algorithm.

### Interface Protocol

XOAI-PMH, with the following additional semantics:

*ListMetadataFormats*

*ODL-Union Results:*

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

*ListSets*

*ODL-Union Results:*

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

*XOAI Response Encoding:*

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

## THE ODL-SEARCH COMPONENT

### Description

One of the most popular services provided by DLs is the ability to perform search operations across one or more of the DL's collections. This service is often integrated into the DL to leverage internal data representations. However,

the search engine can just as easily be abstracted into a component, into which a data stream is fed using the OAI protocol. The ODL-Search component is built upon this philosophy, with queries and results communicated using an extension to the OAI protocol to cast search result sets into OAI sets. Queries may be specified in any language, with an identifier specifying the query language embedded in the request. Figure 3 shows a simple ODL network using ODL-Search.

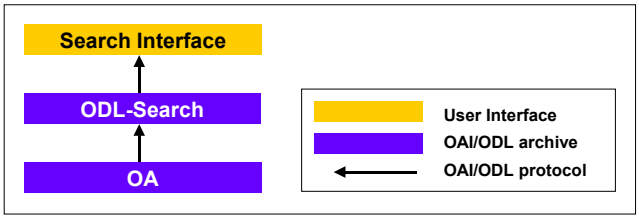


**Figure 3. Simple ODL network using ODL-Search**

### Interface Protocol

XOAI-PMH, with the following additional semantics:

*ListSets*

*ODL-Search Results:*

Empty list.

*ListIdentifiers / ListRecords*

*ODL-Search Parameters:*

qlang
: Name of query language used to indicate the semantics for the query that follows, e.g., odlsearch1.

query
: Search query in language understood by search engine.

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

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

*XOAI Parameter Encoding:*

set
: qlang/query/start/stop

*XOAI Request Encoding:*

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

*Additional ODL-Search Results:*

hits
: Estimated or actual total number of hits.

*XOAI Response Encoding:*

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

### Interoperability Issues

Search engine interoperability is usually equated with the concept of federated or meta-searching – where queries are sent to remote sites, from which results are gathered and merged. In the NDLTD project, Powell and Fox built a system to support federated searching among collections of theses and dissertations [12]. Disparities in search interfaces presented one major obstacle, which can be avoided to some degree by using the ODL-Search protocol. ODL-Search thus can form the basis of a federated system of search engines. Differences in query languages will not be fully accommodated but mapping from one keyword-based query syntax to another is possible.

### Query Language: odlsearch1

*Syntax*

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

*Parameters*

field
: The name of a tag in the original XML data. In the case of Dublin Core records, this could be "title", "creator", or any of the other 13 tags.

term
: A single word, that forms part of the query.

space
: The character used to represent a space.

*Description*

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

*Examples*

- computer science
- +language:ger -description:computer

### Example ODL-Search Requests

```
Verb=ListIdentifiers&set=odlsearch1/computer%20
      science/1/10

Verb=ListRecords&set=odlsearch1/%2Blanguage%3Ag
      er/11/20
```

## EXPERIMENTS

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.

### Methodology

The following components were implemented:

- 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 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

Then, all components were integrated into a simple user interface for the NDLTD system as shown in Figure 4, using metadata corresponding to Electronic Theses and Dissertations [13]. Further, the ODL-Browse component was incorporated into the Computer Science Teaching Center [14].


**Figure 4. User interface for NDLTD ODL**

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 [15]. Thus, instead of response-level containers, such information was embedded into other unused fields in the responses.

### Implementation Details

*Platform*

Each component was built as a separate set of scripts using the Perl language, with data stored in mySQL databases. The implementation used standard Perl modules such as XML::DOM. All coding was done on a Linux platform, with some testing on Digital Unix as well. Wherever possible object-oriented programming was adopted to maximize the use of common components, like modules that assist in harvesting and publishing data through an OAI interface.

*Customization*

The configurable information for each component was stored in an appropriate configuration file, in a well-defined XML format specific to each component. This configuration was used to define the archives from which to harvest data as well as the harvesting parameters, the databases to use and how to access them, as well as any parameters needed for the internal operation of the component. Figure 5 shows an extract of the configuration file for an ODL-Union component.

```
<unionconfig>

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

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

</unionconfig>
```

**Figure 5. Extract from ODL-Union configuration**

### Component Composition

The sequence of interactions corresponding to a typical use of an ODL-Search component is illustrated in Figure 6. The simplified ODL network consists of a source of data in the form of an OAI-compliant archive and an ODL-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.

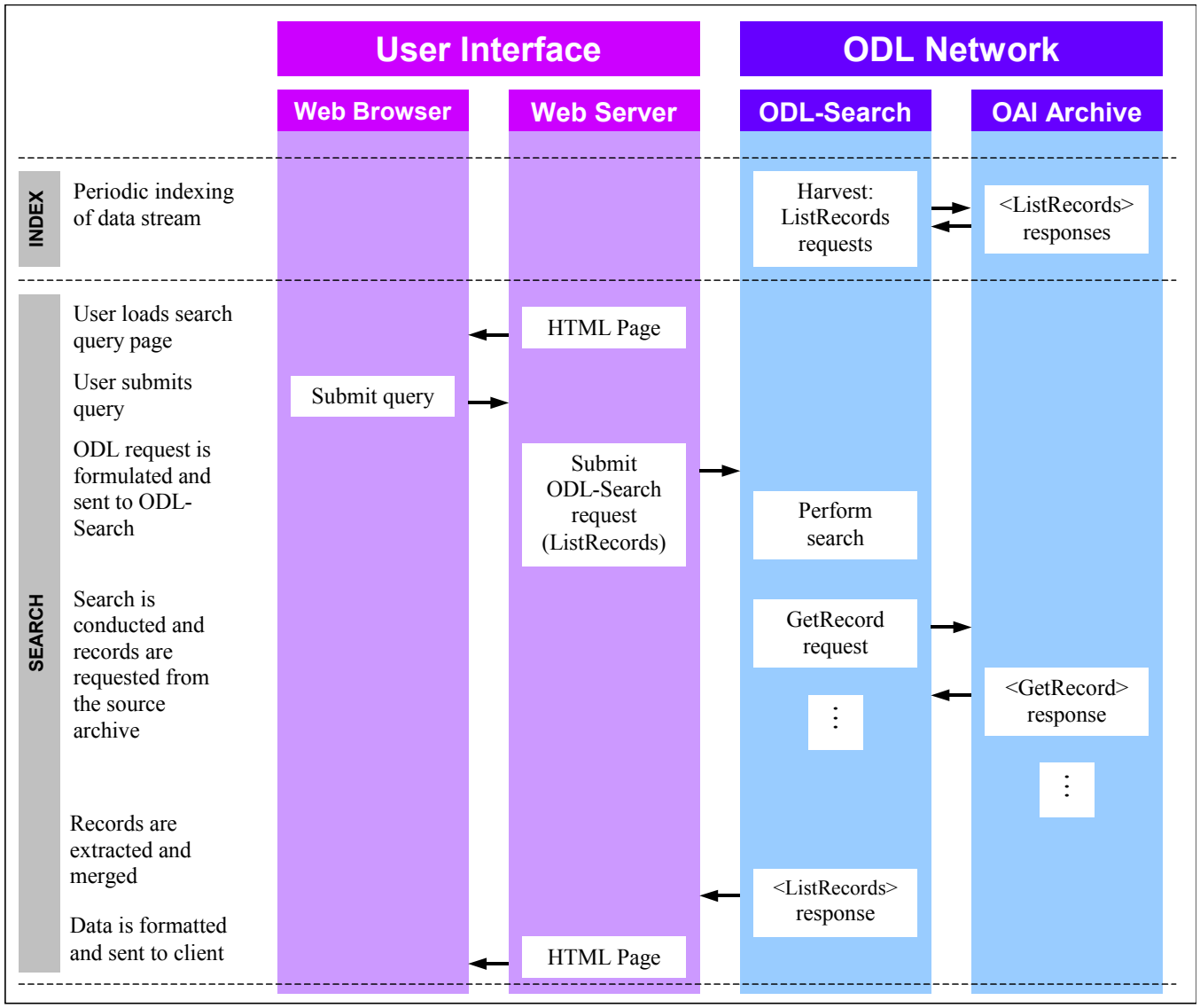


**Figure 6. Interface and component interaction during indexing and search operations**

There are two functions performed: indexing of the data and searching. In the former case, the ODL-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.

To perform a search, the user submits a query by filling in a form on an HTML page. This query is then sent to the Web server, which invokes a script (or handler) to process it. The script extracts the parameters, formulates an ODL-Search ListRecords request and submits this to the ODL-Search component. Upon receiving the request, the ODL-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.

## ANALYSIS

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 explore some of the concerns and related solutions.

### Complexity of Harvesting

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

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

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

Then, the number of batches, $b = \lceil n / k \rceil$

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

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

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

*Harvesting Algorithm A*

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

the number of network requests
= number of batches
= $b$
= $\lceil n / k \rceil$

and the quantity of data transferred
= size of header + (size of record * number of records)
= $h + nR$

*Harvesting Algorithm B*

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

the number of network requests
= number of batches for ListIdentifiers + number of GetRecord requests
= $\lceil n / k \rceil + n$

and the quantity of data transferred
= size of ListIdentifiers header + (size of identifier * number of records) + number of records * (size of GetRecords header + size of record)
= $h + nI + n ( h + R )$
= $( h + nR ) + n ( I + h )$

Thus, the number of network requests as well as the quantity of data transferred will be higher for the second algorithm. The simpler first algorithm was selected for most harvesting operations to avoid this performance penalty.

**Propagation of Change**

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

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

**Duplication of Data**

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

When component A harvests data from a source archive, it has to update all datestamps so that changes will propagate to any downstream components. Thus, component B that harvests from component A will deal with a new system of dates, necessitating an additional overlap of $g$. Compared to the source archive, there will now be a time interval overlap of $2g$ in the records being harvested. Table 3 illustrates this incremental duplication when 2 components are connected in a chain to a source archive and there is an overlap of 1 day. In the illustration items are added at the source, Component A harvests daily from the source, and Component B harvests daily from Component A.

| | **Source**<br>**Item (Date)** | **Component A**<br>**Item (Date)** | **Component B**<br>**Item (Date)** |
|---|---|---|---|
| **Mon** | Item 1 (Mon) | Item 1 (Mon) | Item 1 (Mon) |
| **Tues** | Item 2 (Tues) | Item 1 (Tues)<br>Item 2 (Tues) | Item 1 (Tues)<br>Item 2 (Tues) |
| **Wed** | Item 3 (Wed) | Item 2 (Wed)<br>Item 3 (Wed) | Item 1 (Wed)<br>Item 2 (Wed)<br>Item 3 (Wed) |
| **Thu** | Item 4 (Thu) | Item 3 (Thu)<br>Item 4 (Thu) | Item 2 (Thu)<br>Item 3 (Thu)<br>Item 4 (Thu) |

**Table 3. Illustration of duplication due to overlapping**

Each component in a chain similarly contributes to the overlap in the datestamp range during harvesting. With frequent harvesting and/or frequent updates at the source, this will result in greater duplication of data. Selecting a harvesting granularity that is as small as possible (minimizing $g$) will minimize the duplication due to chaining of components.

**Component Speed Enhancements**

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

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

Then, the number of network requests = $1 + n$

and the quantity of data transferred

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

$= h + nR + n * ( h + R )$

$= h + nh + 2nR$

$= h + n ( h + 2R )$

While the aim of componentization is to make development simpler and repeatable, this cannot be at the expense of reduced functionality or efficiency. The quantities calculated 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 severe penalty was incurred when a list of records needed to be fetched from the ODL-Union component by any of the other components. As illustrated above, the number of additional HTTP requests was proportional to the number of records needed. The most successful and general solutions found to maximize network utilization and minimize the processing delay normally associated with executing Web applications are discussed below.

*Caching*

Using caching at various levels within the experimental systems resulted in significant speed improvements. For example, the ODL-Browse component cached the results from ODL-Union, 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 more of an effect on system performance than most other optimizations. 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.

*FastCGI*

FastCGI 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 and worked very well. 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.

*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 script. This is the approach taken by the SpeedyCGI toolkit, which significantly 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.

## FUTURE WORK AND CONCLUSIONS

### Development and Refinement of Component Libraries

We have developed sample protocol designs for many existing digital library use cases, including threaded discussions, peer review, and 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.

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.

### Component Testing and Validation

Testing of the OAI protocol is largely supported by the Repository Explorer [15], 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.

### 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 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.

### 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 "DL-in-a-box" solutions that can be tailored to classes of applications, such as the National STEM Digital Library (www.nsdl.nsf.gov).

Building upon a 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.


### 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. Van de Sompel, H., and Lagoze, C. The Open Archives Initiative Protocol for Metadata Harvesting. Open Archives Initiative, 2001. Available at http://www.openarchives.org/OAI_protocol/openarchivesprotocol.html.
2. 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.
3. 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 and
4. Lagoze., C., and Davis, J. R. Dienst – An Architecture for Distributed Document Libraries, in *Commun. ACM 38*, 4 (April 1995), 47.
5. Leiner, B. M. The NCSTRL Approach to Open Architecture, in *D-Lib Magazine* (December 1998). Available http://www.dlib.org/dlib/december98/leiner/12leiner.html
6. 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).
7. Birmingham, W. P. An Agent-Based Architecture for Digital Libraries, in *D-Lib Magazine*, (July 1995). Available http://www.dlib.org/dlib/July95/07birmingham.html.
8. 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.
9. Kahn, R., and Wilensky, R. A Framework for Distributed Digital Object Services, CNRI, 1995. Available http://www.cnri.reston.va.us/k-w.html.
10. 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.
11. Dublin Core Metadata Initiative. Dublin Core Metadata Element Set Version 1.1: Reference Description, 1997. Available http://www.dublincore.org/documents/dces/.
12. Powell, J., and Fox, E. A. Multilingual Federated Searching Across Heterogeneous Collections, in *D-Lib Magazine 4*, 8 (September 1998). Available http://www.dlib.org/dlib/september98/powell/09powell.html
13. 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.
14. Computer Science Teaching Center; www.cstc.org/
15. Suleman, H. Enforcing Interoperability with the Open Archives Initiative Repository Explorer, in *Proceedings of JCDL 2001*, (Roanoke, VA, June 2001), ACM Press, 63-64.