SOAPifying the Open Archives

Technical Report Number: CS03-13-00

October 2003

Michael Gaylord
Department of Computer Science
University of Cape Town
mgaylord@cs.uct.ac.za

Sergio Congia
Department of Computer Science
University of Cape Town
scongia@cs.uct.ac.za

Bhavik Merchant
Department of Computer Science
University of Cape Town
bmerchan@cs.uct.ac.za

ABSTRACT

This is a research paper on the partial development and experiments related to a SOAP-compliant metadata harvesting protocol based on the Open Archives Initiative's Protocol for Metadata Harvesting (OAI-PMH). The experiments involved implementations of a client-side Service Provider and a server-side Data Provider and were aimed at determining the feasibility of encoding the OAI-PMH request/response pairs as SOAP messages. In addition, a testing tool was developed to test the protocol compliance of data providers. The results prove that such a SOAP messaging system for the Open Archives is indeed feasible as the additional overhead for such SOAP functionality was insignificantly small. Furthermore, the changes necessary for adopting such a framework are minimal.

1. INTRODUCTION

In recent years the global academic community has begun to rely more and more on what are known as Digital Libraries (DL). A DL is "an electronic information storage system focused on meeting the information seeking needs of its users." [7]

A DL contains metadata records which each describe a logical unit of data that is contained within the library (e.g., a book). A metadata record can essentially be described as "data about data". For example, metadata about a research paper could contain information such as the author, title and year of publication.

A DL in the context of the OAI-PMH is known as a repository. Gathering metadata from a repository is a process known as metadata harvesting, performed by an application known as a harvester. This harvesting is performed so that a client-side program can provide some service to a user (e.g., search facilities). The OAI-PMH is currently the standard for metadata harvesting.

This project investigated the migration of the Open Archives Initiatives Protocol for Metadata Harvesting (OAI-PMH) to a parallel SOAP version. This is due to the recent acceptance of the SOAP Messaging Framework [5] as a World Wide Web Consortium (W3C) endorsed standard for distributed peer-to-peer XML communication over the web.

Previously, no such message-passing standard existed that was ideally suited to the dissemination of metadata. When the OAI began work on the OAI-PMH v2.0 it was already known that the SOAP messaging framework would become a standard in later months. For this reason the OAI-PMH v2.0 was specifically designed so that migration to a SOAP encoding would be possible without changes to the protocol.

The reason for this parallel encoding is simply that of interoperability. The major drive in the OAI's mission is the promotion of interoperability between Data Providers and Service Providers (§ 2.1). A SOAP encoding of the protocol could become a more widely used standard for metadata harvesting. This would be an improvement over the current application specific HTTP GET and POST encodings used for metadata harvesting.

To investigate this aspect, development of specific harvesting tools was needed to support a SOAP version of the OAI-PMH. These tools included a repository, a harvesting tool and a tool to test the compliance of such a repository against the protocol in question. An overview of the message passing between the various components developed can be seen in Figure 1.

SOAP has been implemented in the new encoding as a layer that fits between HTTP and the actual protocol framework. This enables most semantics and syntax of the OAI-PMH version 2.0 [6] to be retained unmodified.

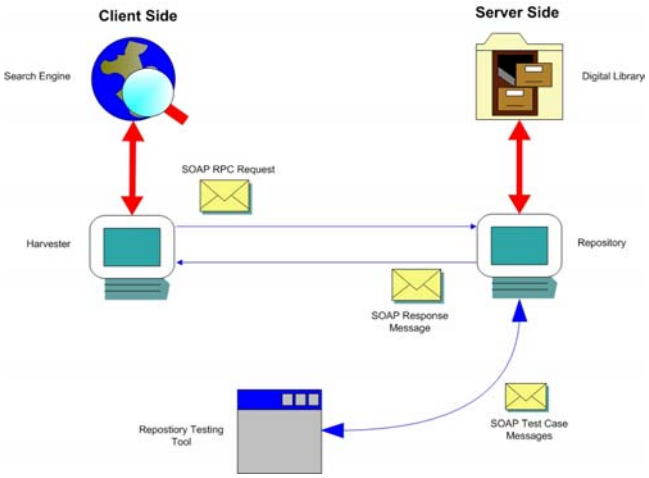

Figure 1: Overview of components developed and their interactions.

2. BACKGROUND & MOTIVATION

2.1 The Open Archives Initiative's Protocol for Metadata Harvesting (OAI-PMH)

Open Archives are information systems that share their data with the outside world using a well-defined application layer Internet protocol. This protocol, developed by the Open Archives Initiative is called the Open Archives Initiative Protocol for

Metadata Harvesting and functions on top of the HTTP transport protocol. The OAI-PMH provides an application-independent interoperable framework based on metadata harvesting. This is a relatively new standard and has been widely adopted by document archives in the research, education and publishing arenas.

The OAI protocol defines an interoperable framework with two classes of participants:

- DATA PROVIDERS - These participants administer systems that support the OAI protocol as a means of exposing metadata about the content in their archives.
- SERVICE PROVIDERS - These participants issue OAI protocol requests to the systems of data providers and use the returned metadata as a basis for building value-added services such as searching, browsing and rights management.

In the OAI-PMH, requests are sent from the Service Provider via a harvester - a Web robot that issues scheduled OAI-PMH requests to the Data Provider. The requests are scheduled regularly to keep the local collections of metadata current. The Data Provider operates a repository that responds to the 6 request types (verbs) defined by the OAI-PMH specification:

- *GetRecord* – used to get a single record from the repository.
- *Identify* – used to obtain archive-level information about the repository.
- *ListIdentifiers* – instructs the repository to return a list of record identifiers according to values given in the request.
- *ListMetadataFormats* – used to query a repository as to the types of metadata formats it supports.
- *ListRecords* – request for a list of all the records specified by the arguments accompanying it.
- *ListSets* – used to retrieve a list of the sets that the repository supports.

2.2 The SOAP Messaging Framework

The SOAP Messaging Framework (SMF) was designed to facilitate the transfer of structured information (XML) over a range of communication protocols [5]. This enables the deployment of standardised Web Service interfaces independent of transport layer protocols.

The structure of a SOAP compliant messaging system comprises a set of SOAP *Producer* and *Consumer* nodes. A node can be defined as a body of programming logic that either relays or processes a SOAP message [5].

As has already been described, the SMF is a specification for using XML documents as messages [4]. The specification contains:

- A model for exchanging SOAP messages.
- A set of rules for representing data within SOAP messages, known as SOAP encoding.
- Guidelines for transporting SOAP messages over HTTP.

2.2.1 The SOAP Message Structure

The SMF is designed to allow simplicity and extensibility. This is facilitated by a fairly loosely defined structure for creating SOAP messages independent of a programming model. The specification defines an element called a *SOAP Envelope* used for data encapsulation. This envelope contains further elements namely:

- A *SOAP Header* – Contains information for exchanging messages in a decentralised manner. This header may contain one or more a *header blocks* whose purpose is to retain information applicable to intermediary nodes along a specific message path.
- A *SOAP Body* – Provides a mechanism for transmitting data. Child nodes in the Body should be namespace qualified. This element essentially contains the payload information of the message.
- A *SOAP Fault* – Used to deliver error information within a SOAP message. The SMF defines semantics for encoding such messages that contain details about the fault such as fault *code*, *reason*, offending *node*, *role* of offending node and *details* about the fault.

Figure 2 shows a SOAP message and its associated components.

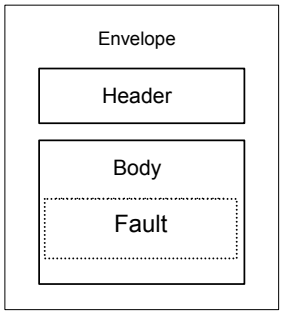

Figure 2: Structure of a SOAP message

3. METHOD

3.1 Protocol Design

In version 2.0 of the OAI-PMH requests are expressed as HTTP URL strings. The valid OAI-PMH URL string consists of the base URL of the repository accompanied by a verb and its associated arguments. Each keyword argument pair is separated by a '&' delimiter. For example, a typical *ListIdentifiers* request sent to the citebase archive with its from, until and metadataPrefix arguments would be:

```
http://citebase.eprints.org/cgi-
bin/oai2?verb=ListIdentifiers&from=2003-01-
01&until=2003-01-01&metadataPrefix=oai_dc
```

The corresponding response to a request is in the form of an XML UTF-8 encoded stream shown in Figure 3 below. (This is a simplified version with namespace declarations and certain other attributes omitted for simplicity)

```
<?xml version="1.0" encoding="UTF-8"?>
<OAI-PMH>
   <responseDate>2003-09-
30T09:43:58Z</responseDate>
   <request metadataPrefix="oai_dc"
verb="ListIdentifiers" until="2003-01-01"
from="2003-01-
01">http://citebase.eprints.org/cgi-
bin/oai2</request>
 <ListIdentifiers>
  <header>
   <identifier>oai:cogprints:2672</identifier>
   <datestamp>2003-01-01</datestamp>
  </header>
  <header>
   <identifier>oai:cogprints:2673</identifier>
   <datestamp>2003-01-01</datestamp>
  </header>
 </ListIdentifiers>
</OAI-PMH>
```

Figure 3: A sample Listidentifiers request

3.1.1 Request Schema

This first step in migrating to a SOAP version was to develop a schema for OAI requests, thus moving from a HTTP POST/GET paradigm to sending an XML encoded request. To avoid redefining types that were already provided in the response schema [3], these types are imported from the OAI-PMH response schema using the import capability of XML schemas.

The request schema[1] defines syntax and semantics of each request verb and is a direct mapping from version 2.0 of the OAI-PMH specification. The corresponding verb arguments, their sequence and types are clearly defined in this schema. Below is an example of the same request found in the earlier example:

```
<?xml version="1.0" encoding="UTF-8"?>
<OAI-PMH-REQ>
   <ListIdentifiers>
      <from>2003-01-01</from>
      <until>2003-01-01</until>
      <metadataPrefix>oai_dc</metadataPrefix>
   </ListIdentifiers>
</OAI-PMH-REQ>
```

Figure 4: A sample ListIdentifiers request

3.1.2 SOAP Encoding

The next step was to embed the newly developed OAI-PMH requests and OAI-PMH responses into a SOAP message. As explained in the previous section a SOAP message contains a body element used to encapsulate payload information. This is the embedding location of OAI-PMH requests and responses. Therefore this process essentially involved making the root element of the request or response, depending on which is to be sent, the first child elements of the SOAP body. Figure 5 and Figure 6 are examples of the same request/response pair as the previous examples encoded as SOAP messages.

[1] The schema can be found at http://simba.cs.uct.ac.za:8180/OAI/SOAP/OAI-PMH-REQ.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope>
 < Body>
  <OAI-PMH-REQ>
   <ListIdentifiers>
    <from>2003-01-01</from>
    <until>2003-01-01</until>
    <metadataPrefix>oai_dc</metadataPrefix>
   </ListIdentifiers>
  </OAI-PMH-REQ>
 </Body>
</Envelope>
```

Figure 5: A sample SOAP-encoded OAI-PMH request

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope>
   <Body>
      <OAI-PMH>
<responseDate>2003-09-
30T09:43:58Z</responseDate>
<request metadataPrefix="oai_dc"
verb="ListIdentifiers" until="2003-01-01"
from="2003-01-
01">http://citebase.eprints.org/cgi-
bin/oai2</request>
      <ListIdentifiers>
       <header>

<identifier>oai:cogprints:2672</identifier>
          <datestamp>2003-01-01</datestamp>
        </header>
        <header>

<identifier>oai:cogprints:2673</identifier>
          <datestamp>2003-01-01</datestamp>
        </header>
      </ListIdentifiers>
    </OAI-PMH>
  </Body>
</Envelope>
```

Figure 6: A sample SOAP-encoded OAI-PMH response

3.2 Implementation

3.2.1 SOAP Data Provider – The Repository

A repository, as has already been discussed, is essentially a database server containing a collection of metadata items. Built on top of the database is a message processing system that controls the dissemination of the metadata.

The implementation of this repository involved four distinct layers. Two were dedicated to the processing of OAI-PMH requests and responses, one involved the backend database while the fourth was used to process SOAP elements.

The repository was implemented in Java and used JDBC to connect to a MySQL database containing a sample collection of electronic theses and dissertations. The Web interface used Java Servlets on Apache Tomcat.

The repository essentially functions in the following manner:

1. The server receives a SOAP message.
2. The SOAP tags are removed from the message and the OAI-PMH request arguments are extracted and processed.

3. The system then makes calls to the database to retrieve the requested information and constructs an OAI-PMH Version 2.0 compliant response.
4. SOAP tags are then attached to the response and it is transmitted to the requesting harvester.

XML is processed in the software with the aid of Apache Xerces – a DOM Level-3 compliant XML parser.

The repository implements the following features:

- OAI-PMH Version 2.0 compliance and the SOAP-OAI-PMH prototype protocol.
- Flow control by way of resumption tokens.
- Selective harvesting using sets.
- Three metadataFormats, namely: oai_dc, marcxml and etdms.
- SOAP processing components are supported as an individual layer.
- Is platform independent and has been tested on both BSD and Microsoft operating systems.
- Uses Apache Xerces to parse and construct XML messages.

3.2.2 SOAP Service Provider – The Harvester

As defined in the OAI-PMH framework, a service provider is an entity which employs the harvesting protocol to obtain metadata from data providers (or metadata repositories). The service provider uses harvested metadata to build value-added services such as searching, browsing, rights management and e-print preservation above the metadata.

The service provider was developed to provide a searching service that allows a user to perform data recollection on harvested metadata. The logic behind this is that the metadata ultimately contains a link to the actual resource it describes. By allowing a user to search metadata, he/she can browse relevant records in an attempt to find the desired resource.

The necessary constituents identified for developing such a service provider were: a harvester – responsible for implementing the harvesting protocol in both HTTP and SOAP forms; a Data Provider Administration tool – used to administer data providers; and a Search engine for information retrieval.

In an attempt to make the system portable, the decision was made to develop the service provider entirely in Java due to the platform independence of Java. Further, to enable the system to be independent of a Database Management System (DBMS), the decision was made to use Lucene (a full featured text-based search engine developed in Java) for storage and indexing of records. Harvested metadata is normalised into a list of records obtained from a data provider by way of a ListRecords request. These records are subsequently added to a Lucene search index which can be queried using a search query.

The service allows users to perform *boolean searches*[2] in both simple and advanced modes. In a simple search, multiple fields of a record are compared and relevant records are displayed in a legible format, while advanced searches are more specific.

3.2.3 Testing Tool

The tool was developed in Java and consisted of a Swing driven GUI. The tool was built in order to test a repository's conformance to the OAI-PMHv2.0 and to the new SOAP encoding of the protocol. The tool was built using standard APIs for Java and is thus easily extensible and platform-independent.

The Testing Tool performs the following major tasks:

- Submits valid individual OAI-PMH requests to the repository (i.e., one of the six request verbs)
  - Informs the user as to what arguments are required for each verb
  - Allows the user to manually modify the parameters of the request
  - Checks user input for errors
- Allows users to select a number of test requests to be run in sequence as a batch
  - Automatically creates and submits requests that are engineered to generate the specific OAI errors chosen by the user
- Displays the responses from the repository as:
  - Raw XML
  - XML formatted to be more human-readable
- Performs validation and error checking, namely:
  - Checks that XML responses are well formed
  - Validates XML responses using online schema
  - Handles any OAI errors gracefully
  - Handles HTTP errors gracefully
  - Handles unexpected errors gracefully (e.g., I/O errors)
- Displays error messages:
  - Individually in the case of single tests
  - Within results tables and log files in the case of batch tests
- Displays the results of batch tests. Namely, for each individual request/response pair from the batch the tool:
  - Shows information about the test case – which verb was sent with the request and what kind of error was expected
  - Shows the result of the test – pass, fail or skipped (if a test is not applicable)
  - Displays the actual request sent or response received
  - Gives the user a reason as to why a test failed
- Gives feedback to the user as to what task the tool is currently performing and its approximate percentage of completion

[2] A Boolean search uses boolean comparisons between words to identify relevant records.

3.3 Evaluation Procedure

The ultimate objective of this project was to develop a prototype protocol that implements the OAI-PMH under a SOAP messaging paradigm. The experimental components implementing this derived protocol were therefore tested together to demonstrate successful functionality. This can be described as complete system integration testing. Integration testing of the complete system was performed by implementing the three experimental components (namely the Data Provider, Service Provider and testing tool) in their full SOAP context and assessing their interoperability.

Another issue to consider was the relative performance of the SOAP-OAI-PMH to the OAI-PMH. This was done to investigate the comparative efficiency between the two protocols. For these tests, message sizes for both requests and responses were compared as well as their respective transfer times.

The final form of testing, protocol conformance testing, was done to ensure that the components of the system proved to support the prototype protocol correctly. To ensure this, independent testing was performed on these components. Two methods of verification were used to test the functions of these components. Since the SOAP-OAI-PMH is a direct mapping of the OAI-PMH, all OAI-PMH related functionality is exactly the same apart from the fact that there is layer responsible for the processing of SOAP related information. The components were thus adapted to support the OAI-PMH version 2.0 and their interoperability was tested with existing OAI-PMH implementations.

The second method of testing protocol conformance of the experimental components involved testing their SOAP functionality. To do this, output of the programs was captured and validated against the SOAP-OAI-PMH schema that was developed (§ 3.1.1). Output was printed to a file and was subsequently analysed using a schema validation tool. This ensured that all information passed between the components conformed to the schema and thus conformed to the protocol.

4. RESULTS

4.1 Results

4.1.1 Complete System Integration Testing

The system functioned as a whole successfully. This means that:

- The service provider was able to selectively harvest all the metadata from the data provider successfully.
- The data provider could service all six possible requests.
- The Testing Tool was able to perform batch tests on the data provider.

4.1.2 Performance Evaluation[3]

Performance evaluation was done with the aid of three specific tests. These tests include:

- *The difference in number of bytes* between a regular OAI-PMH version 2.0 request/response pair and a SOAP-OAI-PMH request/response pair.
- *The difference in transfer time*s between a regular OAI-PMH version 2.0 request/response pair and a SOAP-OAI-PMH request/response pair.
- *The difference in transfer times to harvest the entire contents of the repository* using regular OAI-PMH requests and the SOAP-OAI-PMH.

Evaluation of the message sizes for a specific request, namely a ListRecords request with from, until, metadataPrefix and set arguments, yielded the results found in Table 1.

Table 1: Difference in bytes for a response

Request Type	Request (Bytes)	Response(Bytes)
SOAP	645	167037
HTTP	140	166616
Diff	505	421

The resulting transfer times for the same pair of requests are found in Table 2.

Table 2: Average processing and transfer times for a request

Protocol Version	Processing time (ms)
SOAP	1609.22
HTTP	1594.87
Diff	14.4

The transfer times for harvesting the entire contents of the metadata repository are contained in Table 3 below. This test was performed on a standalone machine using a "dummy" harvesting program. The repository database contained metadata describing 72 376 electronic theses and dissertations.

Table 3: Transfer times for an entire harvest including the average transfer time per request.[4]

	HTTP Version	SOAP Version	Diff
Total Time (ms)	872759	1033584	160825
Avg Time (ms)	361.6	428.3	66.7

4.1.3 Protocol Conformance Testing

The OAI-PMH V2.0 versions of the experimental components were exhaustively tested against other existing OAI-PMH implementations and no problems were encountered. In fact errors were found in existing implementations of OAI-PMHv2.0 repositories.

All the output of these components was captured and found valid against the SOAP-OAI-PMH schema.

[3] These tests were performed using the data provider software module only. A separate set of testing programs was used to issue requests and assess the sizes and transfer times.

[4] Number of requests exercised to harvest the entire database = 2413.

The SOAP Messaging Framework specifies that any XML data contained within the payload of the SOAP envelope must be qualified. Since the SOAP envelope is also qualified the entire message can be validated using schema. Using this approach the testing tool was successfully used to check if the SOAP messages generated by the repository were valid. It was also used to perform batch tests on the repository to ensure that all OAI errors were catered for and indeed they were.

4.2 Discussion of Results

The results for the evaluation of the prototype protocol implementations infer the following:

The proof that the system functioned as a whole successfully meant that the prototype protocol functioned as intended without any unexpected errors. This not only means that the project was a success but also proves that such a "SOAPified" version is a feasible replacement for the OAI-PMH V2.0.

Evaluating the performance of the protocol showed that there were insignificantly small differences for fulfilling SOAP requests as opposed to HTTP requests. The overhead of using the SOAP-OAI-PMH was miniscule (14 ms for a single request and 2.6 minutes to harvest the entire collection of records in the repository). This additional overhead can be attributed to the extra processing required to function with a larger, XML-based request. The overhead in message sizes resulted from the SOAP encapsulation tags and was also deemed insignificant[5].

Since the experimental implementations of the SOAP-OAI-PMH components conformed to the protocol (i.e. all possible requests that each component can produce have been validated), it can be deduced that these components implemented the protocol correctly. Thus any testing between these components can be assumed to have tested the protocol in its entirety.

5. IMPLICATIONS

5.1 The Open Archives Initiative

The SOAP encoding of the protocol was shown to implement all the requirements of the OAI-PMHv2.0 correctly. The fact that the only additional overhead required to implement this new encoding is to encapsulate requests and responses into SOAP envelopes makes it a viable alternative. This encapsulation is intuitive and requires only a marginal amount of additional processing overhead. The OAI is setting up a Working Group to develop a SOAP encoding of the OAI-PMH and it is hoped that the findings of this project help to bring to light issues that need to be addressed during this development.

The results obtained by the members of this project imply that the OAI-PMHv2.0 could indeed be successfully migrated to a new or parallel harvesting protocol.

5.2 SOAP as a Lightweight Messaging Protocol

There has been increased use of SOAP for message passing in the context of Web Services. The successful migration of the OAI-PMHv2.0 to SOAP with very little overhead implies that the design goals of SOAP were successful. It has been shown that SOAP truly is a lightweight messaging protocol.

[5] Less than 56 milliseconds difference with a transfer rate of 8 kilobytes a second (450 / 8000).

6. FUTURE WORK

Although the project was considered successful there are some issues that need to be addressed. A summary is presented below.

- The SOAP specification states that a SOAP "Fault" element must be returned for any error that occurs during message passing. The implementations developed during the life of this project only deal with SOAP faults if the SOAP part of the message is erroneous. Proper conformance to the SOAP specification would mean that SOAP Faults must be returned even for OAI errors.
- The experimental protocol does not deal with SOAP message compression, intermediary nodes and SOAP header blocks.
- Should a standard SOAP encoding be developed the OAI-PMH could be used for metadata dissemination more publicly by allowing Data Providers to publish WSDL information in a UDDI registry. This, however, also depends on WSDL and UDDI being made standards.

7. CONCLUSION

In conclusion, the prototype protocol that was developed for this project and the subsequent experimental implementations of the system that supported it proved that the SOAP Messaging Framework could be considered a viable addition to the OAI-PMH.

After evaluating the performance of the protocol, it has been demonstrated that the overhead introduced for the processing of SOAP messages was insignificantly small and therefore infers that the efficiency of the OAI-PMH version 2.0 protocol was not hampered by the introduction of SOAP elements.

Some investigation still needs to be made into the other features that the SOAP Messaging Framework provides to fully conclude that the experimental protocol that was developed is an alternative to the OAI-PMH version 2.0 protocol.

8. ACKNOWLEDGMENTS

Our thanks to Dr. Hussein Suleman for his much valued guidance as our supervisor for this project

9. REFERENCES

[1] W3C Press Release. *World Wide Web Consortium Issues SOAP Version 1.2 as a W3C Recommendation.* June 2003. [Web Page] Available: http://www.w3.org/2003/06/soap12-pressrelease

[2] Participating Organisations: Ariba, Inc., Commerce One, Inc., Compaq Computer Corporation, DevelopMentor, Inc., Hewlett Packard Company, International Business Machines Corporation, IONA Technologies, Lotus Development Corporation, Microsoft Corporation, SAP AG, UserLand Software, Inc., *SOAP*

Submission Request to W3C [Web Page] Available: http://www.w3.org/Submission/2000/05/

[3] Van de Sompel H., *XML Schema for validating OAI-PMH v2.0 responses*, May 2002. [Online] Available: http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd

[4] Dix C., *SOAP Services*, extract from the book by Cauldwell P., Chawla R., Chopra V., Damschen, G., Dix C., Hong T., Norton, F., Ogbuji, U., Olander G., Richman M.A., Saunders K., and Zaev Z., *Professional XML Web Services* Wrox Press Limited September 2001, ISBN 1861005091 [Web Page] Available: http://www.vbxml.com/soap/articles/soapservices/default.asp

[5] Gudgin M., Hadley M., Mendelsohn N., Moreau J-J., Nielsen H.F., *SOAP Version 1.2 Part 1: Messaging Framwork* W3C Recommendation 24 June 2003 [Web Page] Avaliable at: http://www.w3.org/TR/2003/REC-soap12-part1-20030624/

[6] Lagoze C. Van de Sompel H., *The Open Archives Initiative Protocol for Metadata Harvesting*, Protocol Version 2.0 June 2002 [Web Page] Available: http://www.openarchives.org/OAI/openarchivesprotocol.html

[7] Suleman H., *Open Digital Libraries,* PhD Dissertation, Virginia Tech, November 2002. [Online] Available: http://scholar.lib.vt.edu/theses/available/etd-11222002-155624/unrestricted/odl.pdf