The OAI-PMH Static Repository and Static Repository Gateway

Patrick Hochstenbach
Los Alamos National Laboratory
Research Library, Prototyping Team
Los Alamos, NM 87545-1362
1 (505) 6674448
hochsten@lanl.gov

Henry Jerez
Los Alamos National Laboratory
Research Library, Prototyping Team
Los Alamos, NM 87545-1362
1 (505) 6674448
hjerez@lanl.gov

Herbert Van de Sompel
Los Alamos National Laboratory
Research Library, Prototyping Team
Los Alamos, NM 87545-1362
1 (505) 6674448
herbertv@lanl.gov

ABSTRACT
Although the OAI-PMH specification is focused on making it straightforward for data providers to expose metadata, practice shows that in certain significant situations deployment of OAI-PMH conformant repository software remains problematic. In this paper, we report on research aimed at devising solutions to further lower the barrier to make metadata collections harvestable. We provide an in depth description of an approach in which a data provider makes a metadata collection available as an XML file with a specific format – an OAI Static Repository – which is made OAI-PMH harvestable through the intermediation of software – an OAI Static Repository Gateway - operated by a third party. We describe the properties of both components, and provide insights in our experience with an experimental implementation of a Gateway.

Categories and Subject Descriptors
H.3.7 [**Digital Libraries**]: Standards; System issues

General Terms
Design, Experimentation, Standardization

Keywords
OAI-PMH, metadata harvesting

1. INTRODUCTION
Throughout the different stages that led to the release of version 2 of the Open Archives Protocol for Metadata Harvesting (OAI-PMH) [3, 4, 7, 13, 14, 15], a strong emphasis has been put on devising a specification for metadata harvesting that is straightforward to implement. It is fair to state that, whenever a choice had to be made, the consecutive specifications have favored making it easy for data providers to expose their metadata collections through the protocol instead of for service providers that harvest the exposed metadata. The origin of that bias lies with the Santa Fe Convention of the Open Archives Initiative [7] that aimed at achieving a level of interoperability across repositories of electronic preprints through metadata harvesting.

Recognizing that existing preprint repositories were grass root initiatives operating with quite limited resources, and that new initiatives in that realm would probably operate under similar modest circumstances for some time to come, those involved in the discussions leading to the Santa Fe Convention [13] decided in favor of ease of implementation at the end of the preprint repositories. This strategy was expected to make the barrier to actually exposing metadata through the protocol as low as possible, and eventually increases the impact of preprint-based communication on the scholarly communication system [2].

Nevertheless, for some data providers holding interesting metadata collections, implementation of the protocol has remained problematic. This was first recognized after the release of version 1 of the OAI-PMH, in the context of the Open Language Archives Community (OLAC) project [10]. Several participants in that project wanted to contribute – sometimes small but nevertheless important – metadata collections to the OLAC environment but were unable to do so because of the OAI-PMH-based OLAC strategy. Implementation of the OAI-PMH was not feasible for several OLAC participants, and the reasons ranged from lack of technical expertise, to system administrators having security concerns about operating an OAI-PMH gateway against an enterprise database, to the cost of implementing the protocol being disproportional to the size of the metadata collection to be exposed.

Practice has shown that these problems exist beyond the OLAC Community. In many cases, union catalog projects include participants that are not in a position to operate elaborate software environments, and therefore currently rely on tools such as ftp to add their collection to the central catalog. Also, ideas have been brought forward to trigger duplication of new content in the LoCKSS framework [9] by exposing metadata about that content through the OAI-PMH. It is anticipated that some smaller publishers contributing to the LoCKSS environment will not be able to collaborate in such an OAI-PMH triggered scheme because the technical barrier is too high for them. And, some organizations that are well known in the digital library community make use of web-servers provided by ISPs that do not allow the installation of third party software. Therefore, these organizations cannot share the metadata of their publications through the OAI-PMH.

JCDL '03, May 1-2, 2003, Houston, Texas.

So, it seems that – irrespective of the bias in the OAI-PMH that favors ease of implementation for data providers – the barrier to expose metadata through the OAI-PMH remains too high in certain, non-marginal circumstances. Therefore, we have conducted research to devise an approach that further lowers the barrier to sharing metadata collections through the OAI-PMH.

2. DIRECTIONS EXPLORED

The focus of our research was on delivering an OAI-PMH solution for data providers that are not in a position to operate special software in order to share their metadata collections with harvesters. This focus immediately led to devising solutions by which metadata collections are made accessible as flat files, not databases. And, given that all responses in the OAI-PMH are XML files, this focus narrowed to finding a solution in which a data provider uses an XML file as the container of its metadata collection. Our research led into two quite distinct directions:

- The autonomous data provider approach: In this approach, data providers make an XML file that adheres to an XML Schema created for this purpose available on a Web server, and place an XSL style sheet on that Web server to handle the responses to incoming OAI-PMH requests. Because data providers operating in this mode all use the same format for their XML file, they share a single XSL style sheet. This work led to the insight that, in order to be easily deployable, native support of XSLT in the data provider's Web servers is required. Such support is currently not available by default. Also, experimentation revealed that an implementation of this approach that solely relies on XSLT processing to respond to OAI-PMH requests requires features that are only available in XSL version 2. That specification is currently in a W3C Working Draft status, and conformant tools must be considered experimental. Both insights led us to conclude that, while definitely promising, this track was not mature for actual deployment to our low-barrier target group.

- The dependent data provider approach: In this approach, data providers make an XML file that adheres to an XML Schema created for this purpose available on a Web server, and rely on external, third-party gateway software to make the data from that file harvestable through the OAI-PMH. This track was inspired by the ViDa [8] – Virtual Data Provider – approach introduced by the OLAC Community to remedy the problems described in the Introduction. While the ViDa approach has properties that are specific to the OLAC Community, and was created for version 1 of the OAI-PMH, our research looked for a generic approach to work in conjunction with version 2 of the OAI-PMH. Our work also paid considerable attention to ensuring the accuracy of responses delivered through a gateway to a harvester. Research on this track led to a collaboration with Carl Lagoze, Michael Nelson and Simeon Warner to specify an Implementation Guideline for version 2 of the OAI-PMH. At the time of writing, that Guideline is in its alpha version. When testing of the specification is completed, it will be officially released by the OAI under the name "The OAI Static Repository and Static Repository Gateway" [16]. Research on this track also led to the creation of an experimental gateway. The remainder of this paper reports on both.

3. THE OAI STATIC REPOSITORY MODEL

The OAI Static Repository model provides a simple approach for exposing relatively static and small collections of metadata records through the OAI-PMH. The Static Repository approach is targeted at data providers that have metadata collections ranging in size between 1 and 5000 records and that are not in a position to host OAI-PMH-compliant repository software. However, the model assumes that these data providers do have access to the file services of a standard, network-accessible Web server.

The OAI Static Repository model builds on two types of components:

- The Static Repository - An XML file that is made accessible by a data provider at a persistent network-location. The XML file has a well-defined structure and it contains information similar to that in OAI-PMH responses. This includes metadata records and supporting information required for the purpose of harvesting via the OAI-PMH.

- The Static Repository Gateway – A network accessible server, operated by a third party, that makes one or more Static Repositories harvestable through the OAI-PMH. Due to the fact that a Static Repository Gateway assigns a unique base URL to each such Static Repository, harvesters can harvest Static Repository information in exactly the same manner as they harvest any other OAI-PMH Repository.

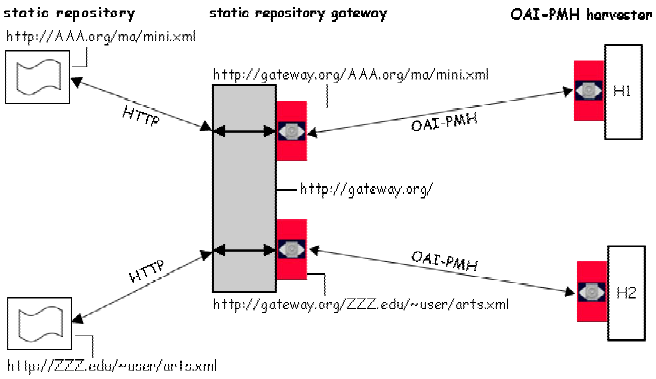

Figure 1. OAI Static Repository Model.

Both the Static Repository and the Static Repository Gateway are described in the remainder of this Section. They are further clarified through Figure 1 and through the example in the Appendix. The full details are available in the OAI Implementation Guideline on OAI Static Repositories and Static Repository Gateways [16].

3.1 The Static Repository

A Static Repository is an XML file that validates against a W3C XML Schema [17] that uses XML elements from the OAI-PMH XML Namespace [18]. The data provider makes the XML file available at a persistent HTTP address. It is anticipated that the data provider will create and update the Static Repository by using an XML editor, or by regularly exporting the status of a metadata collection from a database as a Static Repository XML file. That XML file has sections that contain the responses to the Identify and the ListMetadataFormats OAI-PMH verbs. It also

contains one ListRecords section per Metadata Format supported by the Static Repository.

Taking into account the nature of the environments in which Static Repositories will be created and updated, and aiming for ease of implementation of Static Repository Gateway software, it was decided that Static Repositories can not use optional notions of the OAI-PMH such as "sets", "deleted records" and "seconds-level datestamps".

3.2 The Static Repository Gateway

A Static Repository Gateway (henceforth referred to as Gateway) is a network-accessible server that makes a Static Repository harvestable as an autonomous OAI-PMH repository. In order to achieve this, the Gateway assigns a unique base URL to each Static Repository that it makes harvestable. That base URL is a specific concatenation of the network-location of the Gateway itself, and the HTTP address of the Static Repository. Knowing the specific concatenation rules, data providers can construct the base URL at which a given Gateway will make their Static Repository harvestable. Data providers make their Static Repository known to a Gateway by issuing an OAI-PMH Identify request against the base URL resulting from the concatenation exercise. A Gateway keeps track of all Static Repositories that have "registered" in this manner, and communicates the base URLs of those Static Repositories to harvesters in a Friends [5] container embedded in every Identify response it generates. This allows for dynamic discovery of Static Repositories through a Gateway.

In order to guarantee that harvesters receive adequate information when accessing a Static Repository through a Gateway, the behavior of a Gateway is quite strictly defined. The core rule guiding this behavior is that a Gateway must always use the most recent version of a Static Repository. In theory, this means that a Gateway should fetch a Static Repository from its network-location for every single harvesting request. However, a Gateway can optimize its performance by caching Static Repositories. When caching, a Gateway must perform a freshness-test on the cached Static Repository by comparing it with the version at the Static Repository network-location before responding to harvesting requests. It can do so by using a HTTP HEAD with an If-Modified-Since header that contains the date of the cached version of a Static Repository. Given the above freshness requirements, the following three scenarios can occur:

(1) If the Static Repository is not accessible at its Static Repository network-location when a Gateway performs this freshness-test, it must respond to the harvesting request with a HTTP status-code 504 (Gateway Timeout).

(2) If the Static Repository is accessible at its Static Repository network-location when a Gateway performs this freshness-test, and the freshness-test indicates that the cached version is out-of-date, then it must fetch the Static Repository from its Static Repository network-location:

- If delaying the response until this fetch from the Static Repository is complete and it is processed, the Gateway can respond to the harvesting request with a HTTP status-code 503 (Service Unavailable). This specifies a Retry-After period covering the estimated time of fetching the Static Repository from its Static Repository network-location, and validating it against the Static Repository XML Schema.
- If the fetched version of the Static Repository does not validate against the Static Repository XML Schema, then the Gateway must respond to the harvesting request with a HTTP status-code 502 (Bad Gateway). It must not respond to the harvesting request using the cached version of the Static Repository.
- If the fetched version of the Static Repository does validate against the Static Repository XML Schema, then the Gateway must respond to the harvesting request using the fetched version.

(3) If the Static Repository is accessible at its Static Repository network-location when a Gateway performs this freshness-test, and the result of the freshness-test indicates that the cached version is the same as the version at the Static Repository network-location, then the Gateway may respond to the harvesting request by using the cached version of the Static Repository.

4. A GATEWAY IMPLEMENTATION

As described in the Introduction, the aim of the Static Repository specification is to make participation in an OAI-PMH harvesting environment easier for data providers. This is achieved by allowing data providers to put metadata collections out as XML files that adhere to a well-defined format. Data providers then rely on the services of a Gateway to make the information in such XML files harvestable through the OAI-PMH. Especially due to the strictly defined behavior of Gateways imposed to ensure accuracy of harvested data, the implementation of conformant Gateway software seems not trivial. We set out to create experimental Gateway software, to check the feasibility of the OAI Static Repository specification, and – by sharing our experiences in doing so through this paper – to motivate third parties to create robust Gateway implementations.

Our Gateway approach builds on four components:

- The OAI-PMH Interface – A CGI program that accepts OAI-PMH requests targeted at Static Repositories; performs the freshness-test of Cached Static Repositories for incoming OAI-PMH requests; delivers OAI-PMH responses in case a Cached Static Repository was determined to be fresh; generates the appropriate HTTP status-codes when the freshness-test failed; and communicates the necessity of updating a Cached version to the Daemon through the Lock Zone.
- The Cache – A file-system based storage space in which Cached versions of individual Static Repositories are held as separate GDBM databases [1].
- The Lock Zone - A file-system based storage space that acts as a serving-hatch between the OAI-PMH Interface and the Daemon. It holds Lock Files, each of which contain information on a Static Repository that needs to be fetched as a result of a failed freshness-test, as well as on the actual status of the fetching process.
- The Daemon – A daemon that continuously monitors the Lock Zone; fetches Static Repositories when the Lock Zone indicates that doing so is required; updates the status of the fetching process in the Lock Files; updates the Cache.

The remainder of this Section describes these components and their interaction in more detail. That description is further supported by Figure 2.

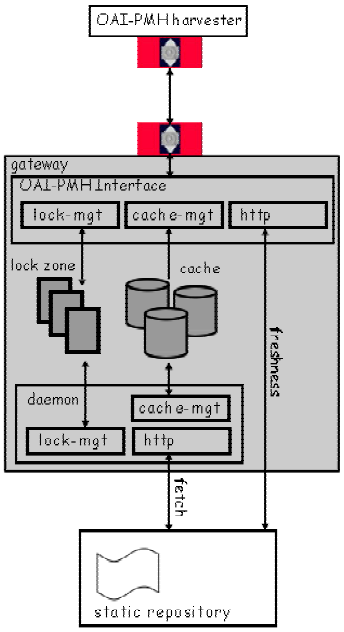

Figure 2. A Static Repository Gateway Implementation.

4.1 The OAI-PMH Interface

The OAI-PMH Interface consists of a front-end that ingest OAI-PMH requests, checks those for syntactic validity and responds with appropriate error messages in case requests are invalid. It also passes on responses delivered to it by the back-end of the OAI-PMH Interface, which in itself consists of three components that are called in the listed order:

- The Lock Management Component – Writes information on Static Repositories for which the Cached version is out-of-date to the Lock Zone.
- The Cache Management Component - Interacts with the Cached Static Repositories.
- The HTTP Component – Performs the freshness-test of Cached Static Repositories.

Valid incoming OAI-PMH requests targeted at a specific Static Repository are initially handed over to the Lock Management Component that checks whether a process of caching the Static Repository is currently ongoing, and if so what the status of that process is.

- In case such a process is indeed ongoing, the front-end of the OAI-PMH Interface responds to the harvesting request with an HTTP status-code of 503 (Service Unavailable) specifying a Retry-After period. The Lock Management Component can derive such status information from the appropriate Lock File in the Lock Zone.
- If no such process is ongoing, control is handed over to the Cache Management Component of the OAI-PMH Interface.

Using a unique key derived from the HTTP address of the targeted Static Repository as the entry into the Cache, the Cache Management Component checks for the existence of a Cached version of the Static Repository. The following two scenarios can occur:

(1) If such a Cached version exists, then the Cache Management Component checks the date/time of the Cached version of that Static Repository. Next, the HTTP Component issues an If-Modified-Since HTTP HEAD request using the obtained date/time against the HTTP address of the Static Repository.

- If doing so reveals that the Cache is fresh, the Cache Management Component reads the appropriate information from the Cached GDBM database for the Static Repository, and hands that information over to the front-end of the OAI-PMH Interface, which can then respond to OAI-PMH request. Depending on whether the Cache indicates that Static Repository is a valid or invalid, the response will be a regular OAI-PMH response containing data, or an HTTP status-code 502 (Bad Gateway).
- If doing so reveals that the Cached version is out-of-date, the Lock Management Component writes a Lock File in the Lock Zone specifying the HTTP address of the Static Repository that needs updating as well as the current status of this fetch, which at this point is "unprocessed". Also, the front-end responds with an HTTP status-code 503 (Service Unavailable), specifying a Retry-After period that is a best guess of the amount of time it may take to update the Cached version. At this point, from the perspective of the Gateway, the OAI-PMH request has been processed. The harvester will need to re-issue the request after the Retry-After period, in order to receive an OAI-PMH response that contains actual data.
- If doing so is unsuccessful in that there is no response to the If-Modified-Since HTTP HEAD request, then the front-end responds with an HTTP status-code 504 (Gateway Timeout).

(2) If such a Cached version does not yet exist, the Lock Management Component writes a Lock File, and the front-end responds with an HTTP status-code of 503 (Service Unavailable) specifying a Retry-After period.

4.2 The Cache

The Cache consists of individual GDBM databases, one per Cached Static Repository. The filename of each GDBM database is a unique key derived from the HTTP address of the Static Repository. Its content consists of administrative information such as date/time of first and most recent caching of the Static Repository, and a processed version of the Static Repository that makes responding to OAI-PMH requests a matter of simply joining appropriate portions of stored XML data obtained by deconstructing the Static Repository XML file.

4.3 The Lock Zone

The Lock Zone is read/write accessible by both the Lock Management Component of the OAI-PMH Interface and the Daemon. When the freshness-test of a Cached version of a Static Repository reveals that the Cached version is out-of-date or not yet existing, the Lock Management Component writes a Lock File in the Lock Zone stating the HTTP address of that Static Repository, its name in the Cache, as well as the "unprocessed"

status of the process of updating the Cached version. The Lock Zone is monitored by the Daemon, which interprets a Lock File as an instruction to fetch a Static Repository from its HTTP address. As will be explained in the following Section, the Daemon updates the status of a file in the Lock Zone as it acts upon the fetching instruction; it eventually removes the Lock File from the Lock Zone.

4.4 The Daemon

The Daemon continuously monitors the Lock Zone and acts upon the Lock Files deposited there by the Lock Management Component of the OAI-PMH Interface. The Daemon itself consists of three components that are called in the listed order:

- The Lock Management Component – Reads Lock Files with "unprocessed" status; updates status information of Lock Files as the process of updating/writing the Cached version of the corresponding Static Repository is ongoing; eventually removes Lock Files from the Lock Zone.
- The HTTP Component – Fetches Static Repositories from their network-location.
- The Cache Management Component – Replaces the out-of-date Cached version of a Static Repository by the newly fetched version or creates a Cached version if no Cached version exists; writes a flag if the newly fetched version is not a valid Static Repository.

The Daemon interprets each individual Lock File with a status of "unprocessed" as an instruction to cache a fresh version of the associated Static Repository. The refreshing process starts with the Daemon attempting to fetch the Static Repository from its HTTP address.

- If fetching fails, the Daemon deletes the Lock File. The Cached version will remain out-of-date, and as a result the freshness-test will fail again when the harvester re-issues the OAI-PMH request after the Retry-After period. The process described in Section 4.1 will start from scratch. Eventually, the harvester may decide to give up, or the Static Repository may become accessible. It can be anticipated that the Gateway would maintain the fetching history of Static Repositories, and decide to remove some from its Cache and Friends list based on a history that reveals an unacceptable level of inaccessibility.
- If fetching is successful, the Daemon proceeds to validating the fetched Static Repository. During the validation process, the Daemon updates the status of the Lock File at several points. If the fetched file is a valid Static Repository, its content is used to replace the existing Cached version. The date/time of most recent caching is updated. If no Cached version exists yet, it is created, and the date/time of first and most recent caching is recorded. After doing so, the Daemon removes the Lock File from the Lock Zone. When the harvester returns after the Retry-After period, it is most likely that a response can be generated from the Cache, since chances are high that the freshness-test to be performed for the re-issued request will reveal that the Cached version is still up-to-date. If the fetched file turns out not to be a valid Static Repository, a flag is set in the GDBM database for that Static Repository. Again, the date/time of most recent caching is updated. If no Cached version exists for the fetched invalid Static Repository, it is created. Its only content will be the "invalid" flag, and the date/time of first and most recent caching. Once the "invalid" flag is recorded, the Daemon removes the Lock File from the Lock Zone. When the harvester returns after the Retry-After period, a HTTP status-code 502 (Bad Gateway) response can most likely be generated based on the existing invalid flag in the Cache, since chances are high that the freshness-test to be performed for the re-issued request will reveal that the invalid Cached version is still the up-to-date version of the Static Repository.

5. DISCUSSION

Static Repositories made available through our Gateway pass the validation tests of both the OAI Repository Explorer [11, 12] and the OAI Registry [6].

The current implementation takes some basic precautions inspired by the security considerations listed in the Static Repository specification [16]. For example, an upper limit is imposed on the total amount of Static Repositories that can be Cached and processed at a given point in time, on the size of Cached Static Repositories, as well as on the size of responses sent to harvesters.

In order to guarantee accuracy of responses to harvesting requests our implementation has paid special attention to the actual implementation of the freshness-test. Web servers on which Static Repositories are made available may operate in other time zones than the Gateway, and are not necessarily synchronized to an Internet time-server. Therefore, using the Gateway's time when issuing an If-Modified-Since HTTP HEAD request may lead to significant inaccuracy of the freshness-test. In order to resolve this problem, our implementation stores the content of the Web server's Last-Modified header field in the GDBM database of the Static Repository, and uses that information in a subsequent freshness-test. As such, the freshness-test is always performed according to the Web server's time.

Our Gateway implementation was written in C and tested on a 500 Mhz Redhat Linux 7.3. Processing and Caching fetched Static Repositories takes between 1 second for a small XML file and 5 seconds for files that reach our upper limit of 2 Mb. Little robust information can be given on the time required to fetch Static Repositories, as those are dependent on the size of the XML file, and are subject to network conditions. In our testing environment, performing freshness-tests has typically taken between one and two seconds. The time to perform a freshness-test is relevant in that it is good indication of the maximum amount of time a harvester must wait for a response to an OAI-PMH request:

- If a freshness-test reveals that the Cache is still up-to-date, generating a response from Cache requires a little extra time due to the deconstructed manner in which Static Repositories are Cached.
- If a freshness-test reveals that the Cache is out-of-date the HTTP status-code of 503 (Service Unavailable) can be sent immediately.

The only occasion at which responding to a harvester takes longer is when the Web server on which the Static Repository is available fails to respond. Our implementation generates an

HTTP status-code of 504 (Gateway Timeout) after having waited for 30 seconds.

6. CONCLUSIONS

Our research into devising an approach to further lower the barrier for data providers to share metadata collections in an OAI-PMH environment led us into two directions. Both directions are based on the data provider making its metadata collection available on a Web server as an XML file of a specific format.

In the "dependent data provider approach" detailed in this paper, data providers rely on the services of a gateway operated by a third party to make metadata collections harvestable. The barrier for sharing data via the OAI-PMH is lowered significantly in that the task of data providers consists of creating and updating an XML file containing their metadata records, placing the file on a Web server and "registering" it with a Static Repository Gateway. This approach depends on the actual deployment of such Gateways. In order to guarantee accuracy of the data harvested through a Gateway the specification of its behavior is quite strict, and therefore adequate care must be taken when creating an actual Gateway implementation. Nevertheless, our experiment revealed that no significant hurdles are involved in an actual implementation that could keep parties from stepping forward to create and deploy robust Gateway software.

At the time of writing, both the OAI Implementation Guideline on Static Repositories and our Gateway implementation are in alpha phase, with feedback on both being gathered from selected parties. Based on the attention our work has attracted so far, it is anticipated that parties that are likely to start exposing metadata via a Static Repository approach will emerge in a variety of communities. The OLAC Community has indicated interest in migrating to the generic Static Repository approach; union catalog projects in Belgium, Brazil, and the United States are considering adoption; and institutions collaborating with the Digital Library Federation and the National Science Digital Library project are exploring the use of this low-barrier approach as a means to significantly increase the amount of metadata records they make harvestable at limited expense.

In the "autonomous data provider approach" on which this paper only briefly touches, data providers use an XSL style sheet – which could be provided by the OAI – to respond to OAI-PMH requests. Their task consists of creating and updating an XML file containing their metadata records, and placing both the XML file and the XSL style sheet on their Web server. Not only does this approach significantly lower the barrier for sharing metadata collections through the OAI-PMH, it also turns the target group of low-barrier data providers into autonomous operators of OAI repositories. While truly promising, we decided that this approach was not ready for deployment to our target group due to the status of technologies required in the solution. Deployment may however become feasible and attractive in the near future.

7. ACKNOWLEDGMENTS

The authors wish to thank Carl Lagoze, Michael Nelson, and Simeon Warner for invaluable input in the process of specifying the OAI Static Repository Implementation Guideline. The authors are grateful for the most inspiring ViDa work of Steven Bird and Gary Simons on behalf of the OLAC Community. Thanks to Beth Goldsmith, Rick Luce, and Thorsten Schwander for feedback.

8. REFERENCES

[1] Free Software Foundation. GDBM. http://www.gnu.org/software/gdbm/gdbm.html

[2] Ginsparg, P., Luce, R., and Van de Sompel, H. The Open Archives Initiative aimed at the further promotion of author self-archived solutions, 1999. http://www.openarchives.org/meetings/SantaFe1999/ups-invitation-ori.htm

[3] Lagoze, C. and Van de Sompel, H. The Open Archives Initiative: Building a low-barrier interoperability framework… in Proceedings on ACM/IEEE Joint Conference on Digital Libraries (Roanoke VA, June 2001), ACM Press, 54-62. http://doi.acm.org/10.1145/379437.379449

[4] Lagoze, C., Van de Sompel, H., Nelson, M., and Warner, S. The Open Archives Initiative Protocol for Metadata Harvesting - Version 2.0, 2002 http://www.openarchives.org/OAI_protocol/openarchivesprotocol.html

[5] Lagoze, C., Van de Sompel, H., Nelson, M., and Warner, S. Implementation Guildelines for the Open Archvies Initiative for Metadata Harvesting: XML Schema for repositories to list confederate repositories, 2002 http://www.openarchives.org/OAI/2.0/guidelines-friends.htm

[6] The Open Archives Initiative. Registering as a Data Provider. http://www.openarchives.org/data/registerasprovider.html

[7] The Open Archives Initiative. The Santa Fe Convention, 2001. http://www.openarchives.org/sfc/sfc_entry.htm

[8] The Open Language Archives Community. How to become an OLAC data provider. http://www.language-archives.org/docs/implement.html

[9] Reich, V. and Rosenthal D. LOCKSS: A Permanent Web Publishing and Access System. D-Lib Magazine, 7 (6), 2001. http://www.dlib.org/dlib/june01/reich/06reich.html

[10] Simons, G. and Bird, S. Building an Open Language Archives Community on the OAI Foundation, 2003. Library Hi Tech, 21(2). To appear.

[11] Suleman H. Enforcing interoperability with the open archives initiative repository explorer… in Proceedings on ACM/IEEE Joint Conference on Digital Libraries (Roanoke VA, June 2001), ACM Press, 63-64. http://doi.acm.org/10.1145/379437.379450

[12] Suleman H. The OAI-PMH Repository Explorer. http://www.purl.org/NET/oai_explorer

[13] Van de Sompel, H. and Lagoze, C. The Santa Fe Convention of the Open Archives Initiative. D-Lib Magazine, 6 (2), 2000. http://www.dlib.org/dlib/february00/vandesompel-oai/02vandesompel-oai.html

[14] Van de Sompel, H. and Lagoze, C. The Open Archives Initiative Protocol for Metadata Harvesting - Version 1.0, 2001. http://www.openarchives.org/OAI/1.0/openarchivesprotocol.htm

[15] Van de Sompel, H. and Lagoze, C. Notes from the Interoperability Front: A Progress Report from the Open Archives Initiative. Lecture Notes in Computer Science, 2458: Proceedings of ECDL 2002 (Rome Italy, September 2002), Springer Verlag, 144-157

[16] Van de Sompel, H., Lagoze, C., Nelson, M., and Warner, S. Implementation Guidelines for the Open Archives Initiative for Metadata Harvesting: The OAI Static Repository and Static Repository Gateway, 2002 http://www.openarchives.org/OAI/2.0/guidelines-static-repository.htm

[17] Van de Sompel, H. and Jerez, H. XML Schema defining the OAI Static Repository format, 2002 http://www.openarchives.org/OAI/2.0/static-repository.xsd

[18] Van de Sompel, H. XML Schema for validating responses to OAI-PMH requests, 2002 http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd

9. Appendix

Table 1 shows an OAI Static Repository, which supports two Metadata Formats (oai_dc and oai_rfc1807). It contains metadata about a single resource. That metadata is provided in both Metadata Formats. Note the metadataPrefix attribute that extends the ListRecords element from the OAI-PMH XML Namespace [18]. To improve readability, XML Namespace declarations are not shown in the sample Static Repository.

```
<?xml version="1.0" encoding="UTF-8"?>
<Repository>
 <Identify>
  <oai:repositoryName>Demo</oai:repositoryName>
 <oai:baseURL>http://an.oai.org/ma/mini.xml</oai:baseURL>
  <oai:protocolVersion>2.0</oai:protocolVersion>
  <oai:adminEmail>jondoe@oai.org</oai:adminEmail>
  <oai:earliestDatestamp>2002-09-19</oai:earliestDatestamp>
  <oai:deletedRecord>no</oai:deletedRecord>
  <oai:granularity>YYYY-MM-DD</oai:granularity>
 </Identify>
 <ListMetadataFormats>
  <oai:metadataFormat>
   <oai:metadataPrefix>oai_dc</oai:metadataPrefix>
   <oai:schema>
       http://www.openarchives.org/OAI/2.0/oai_dc.xsd
   </oai:schema>
   <oai:metadataNamespace>
      http://www.openarchives.org/OAI/2.0/oai_dc/
   </oai:metadataNamespace>
  </oai:metadataFormat>
<oai:metadataFormat>
   <oai:metadataPrefix>oai_rfc1807</oai:metadataPrefix>
   <oai:schema>
       http://www.openarchives.org/OAI/1.1/rfc1807.xsd
   </oai:schema>
   <oai:metadataNamespace>
       http://info.internet.isi.edu:80/in-notes/rfc/files/rfc1807.txt
   </oai:metadataNamespace>
  </oai:metadataFormat>
 </ListMetadataFormats>
 <ListRecords metadataPrefix="oai_dc">
  <oai:record>
   <oai:header>
    <oai:identifier>oai:an.oai.org:0112017</oai:identifier>
    <oai:datestamp>2003-01-17</oai:datestamp>
   </oai:header>
   <oai:metadata>
    <oai_dc:dc>
     <dc:title>Structural Metadata</dc:title>
     <dc:creator>Smith, Hector</dc:creator>
     <dc:subject>Digital Libraries</dc:subject>
     <dc:date>2001-12-14</dc:date>
    </oai_dc:dc>
   </oai:metadata>
  </oai:record>
 </ListRecords>
 <ListRecords metadataPrefix="oai_rfc1807">
  <oai:record>
   <oai:header>
    <oai:identifier>oai:an.oai.org:0112017</oai:identifier>
    <oai:datestamp>2002-01-15</oai:datestamp>
   </oai:header>
   <oai:metadata>
    <oai_rfc1897:rfc1807>
        <rfc1807:bib-version>v2</rfc1807:bib-version>
        <rfc1807:id>0112017</rfc1807:id>
        <rfc1807:entry>January 15, 2002</rfc1807:entry>
        <rfc1807:title>Structural Metadata</ rfc1807:title>
        <rfc1807:author>Hector Smith</rfc1807:author>
        <rfc1807:date>December 14, 2001</rfc1807:date>
    </oai_rfc1897:rfc1807>
   </oai:metadata>
  </oai:record>
 </ListRecords>
</Repository>
```

Table 1: An OAI Static Repository