The multi-faceted use of the OAI-PMH in the LANL Repository

Henry N. Jerez hjerez@lanl.gov
Xiaoming Liu liu_x@lanl.gov
Patrick Hochstenbach hochsten@lanl.gov
Herbert Van de Sompel herbertv@lanl.gov
Digital Library Research & Prototyping Team
Los Alamos National Laboratory, Research Library
Los Alamos, New Mexico, USA

ABSTRACT
This paper focuses on the multifaceted use of the OAI-PMH in a repository architecture designed to store digital assets at the Research Library of the Los Alamos National Laboratory (LANL), and to make the stored assets available in a uniform way to various downstream applications. In the architecture, the MPEG-21 Digital Item Declaration Language is used as the XML-based format to represent complex digital objects. Upon ingestion, these objects are stored in a multitude of autonomous OAI-PMH repositories. An OAI-PMH compliant Repository Index keeps track of the creation and location of all those repositories, whereas an Identifier Resolver keeps track of the location of individual objects. An OAI-PMH Federator is introduced as a single-point-of-access to downstream harvesters. It hides the complexity of the environment to those harvesters, and allows them to obtain transformations of stored objects. While the proposed architecture is described in the context of the LANL library, the paper will also touch on its more general applicability.

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

General Terms
Design, Standardization.

Keywords
Digital Libraries, OAI-PMH, interoperability, federation.

1. INTRODUCTION
When compared to most academic and research libraries, the Research Library of the Los Alamos National Laboratory (LANL) follows a rather unique strategy with respect to providing access to electronic scholarly information. The general trend in electronic library services is to have users access externally hosted materials through third party services, federated through a locally hosted Web Portal. In order to be self-supporting with respect to mission-critical scholarly information, the LANL library acquires or licenses a vast collection of digital scholarly assets, hosts those assets locally, and makes them accessible through locally developed user services. The locally hosted assets include secondary data feeds from ISI, BIOSIS, Inspec, and primary information feeds from major scholarly publishers such as Elsevier, Wiley, IOP, APS, etc.

At the time of writing the collection of locally hosted assets amounts to approximately 8 Terabytes of raw materials. In addition to that, the LANL library is actively investigating the deployment of Institutional Repository capabilities to host locally created materials such as technical reports, datasets, videotaped presentations, etc. Also, research is underway to augment the locally hosted collection with materials gathered by focused Web crawling, and to include logs detailing the usage of repository assets in the repository as assets in their own right [1,2]. Hosting, archiving and making accessible such a vast and heterogeneous collection of scholarly assets in a consistent and sustainable manner is a challenge that touches on many areas of Digital Library practice and research, including the identification of assets, the expression of relationships between assets, the representation of assets by means of complex object models, and methods to ingest and access stored assets.

Over the last year, the Digital Library Research and Prototyping Team of the LANL Research Library has worked on the design of a LANL Repository architecture aimed at ingesting, storing, and making accessible to downstream applications its ever growing heterogeneous digital collection. Also, a working prototype of the design has been implemented. While no claims are being made that the LANL Repository design or implementation are of a nature that merits a comparison with – say – the deliverables of the DSpace [3] or Fedora [4] projects, the authors do feel that the architecture has the following interesting properties that should be attractive for repository-related projects beyond the realm of the LANL Research Library:

a. The use of the MPEG-21 Digital Item Declaration Language (DIDL) to represent complex objects, as described in [5].
b. The natively distributed nature of the architecture.
c. The use of a special technique – the XMLtape – to store and make accessible static collections of complex objects.
d. The multi-faceted use of the OAI-PMH to access stored content in incremental batches.
e. The use of NISO OpenURL to access stored content or various disseminations thereof, as described in [6].

JCDL '04, June 7–11, 2004,Tucson,Az,USA.

f. The dynamic binding of dissemination methods to stored content, as described in [6].

This paper elaborates on properties (b), (c), (d) of the LANL Repository architecture by describing its multi-faceted use of the OAI-PMH [7]. Generally speaking, the OAI-PMH is used in the architecture to enable downstream applications – such as indexing engines of various types – to recurrently poll the LANL Repository for added assets that are of interest to them, to harvest them, and – in good OAI-PMH Service Provider tradition – to do something meaningful with them. In order to achieve this, the OAI-PMH is used at different levels in the architecture. This will be shown in the remainder of this paper by describing the major components used in the architecture, as well as their interactions. A good insight in the OAI-PMH is required for an adequate understanding of this paper. To improve calrity, terms from the OAI-PMH are showed in `another font`.

Figure 1 introduces those major components. As can be seen, the LANL Repository hosts a multitude of autonomous OAI-PMH repositories, each of which stores complex digital objects represented using an XML wrapper format. This aspect of the LANL Repository will be discussed in Sections 2 and 3. The Repository Index, detailed in Section 4, keeps track of the creation of such autonomous OAI-PMH repositories as well as of their location. The Repository Index itself is exposed as an OAI-PMH repository in its own right. For each complex digital object stored in the environment, the Identifier Resolver stores the identifier of that object as well as the location of the OAI-PMH repository in which it resides. The Identifier Resolver, described in Section 5, is populated through OAI-PMH harvesting and can be queried in a variety of ways, including the handle protocol. The DIP engine, which works according to MPEG-21 principles, is introduced to facilitate the delivery of various disseminations of stored objects. The DIP engine is only briefly described in Section 6; details are available in [5, 6]. Finally, the OAI-PMH Federator, detailed in Section 6, exposes the whole LANL Repository as a single OAI-PMH repository. It interacts with all other components mainly using the OAI-PMH. The OAI-PMH Federator hides the complexity of the environment to downstream harvesters, and becomes their single point of access to harvest from the LANL Repository. The OpenURL Gateway is described in detail in [6]; it provides a front-end to the repository from which various disseminations of individual objects contained in the LANL Repository can be obtained using requests compliant with the forthcoming NISO OpenURL standard [8].

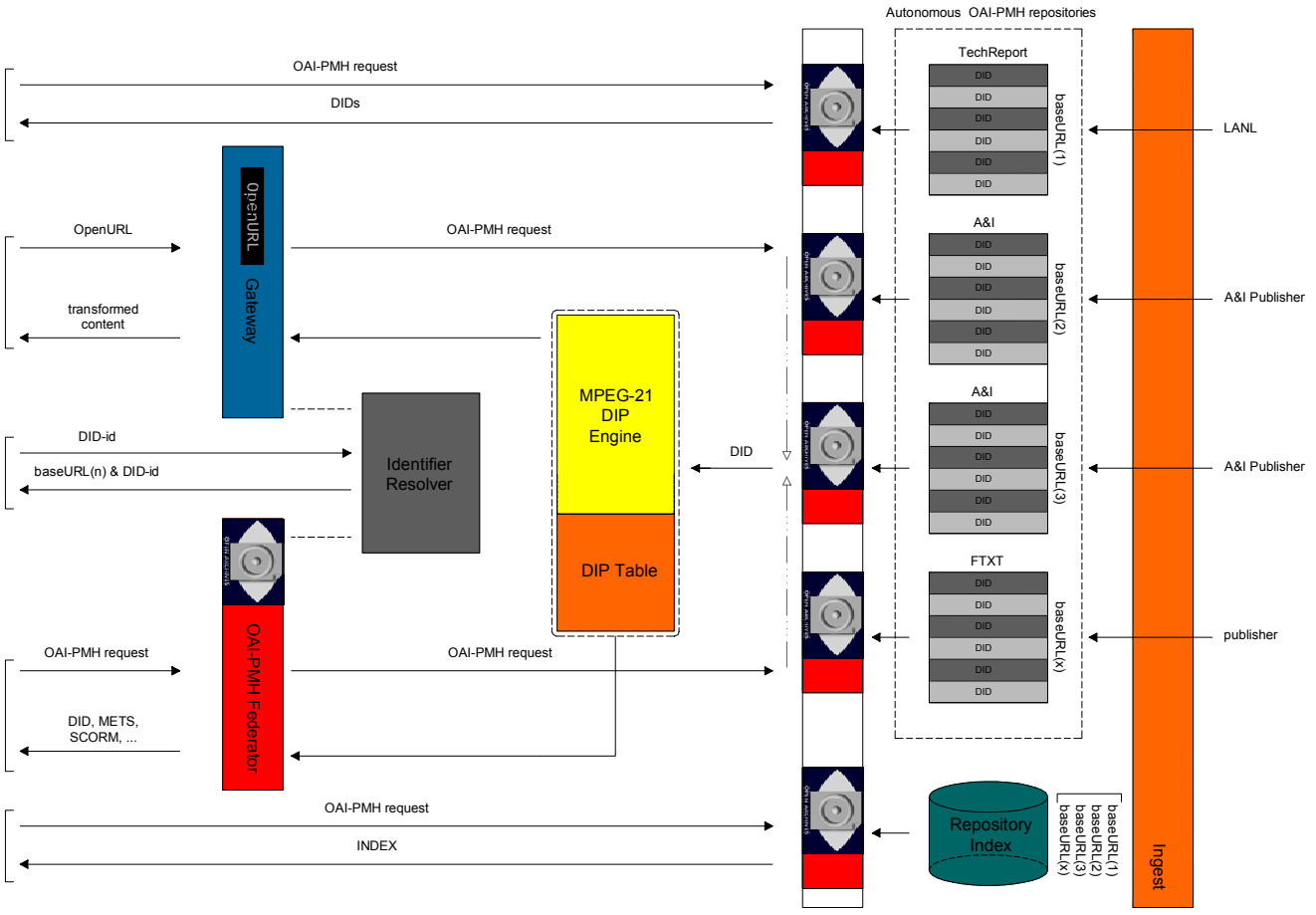

Figure 1. LANL Repository Architecture

2. INGESTION INTO THE LANL REPOSITORY

In many cases the delivered assets to be hosted in the LANL Repository are 'complex' in the sense that they consist of multiple individual datastreams that form a single logical unit. For example, a scholarly article may be delivered as a bundle that consists of metadata describing the article, the article itself in PDF and ASCI format, and the references made in the article expressed in XML. The complex nature of the assets led to an investigation regarding existing approaches to represent complex digital objects using XML wrappers, which resulted in the selection of the MPEG-21 Digital Item Declaration Language (DIDL) [9] as the sole way to store digital assets in the LANL Repository. The actual use of DIDL in the LANL Repository, including the dynamic manner in which dissemination methods are attached to assets upon retrieval, is described in detail in [5,6].

Digital assets to be hosted by the LANL Repository can, in principle, be obtained in a variety of ways including ftp, OAI-PMH harvesting, Web crawling and delivery on physical media. A prototype ingestion process has been developed that turns each obtained asset into an autonomous XML document that wraps the datastream(s) of which the asset consists. Such an XML document is named a Digital Item Declaration (DID); all LANL DIDs are compliant with the MPEG-21 DIDL specification. As such, for example, the different datastreams of the previously mentioned scholarly article will be contained in a single DID, which will physically contain and/or reference the various datastreams that make up the asset. The DID also contains information added by the ingestion process, for example, aimed at expressing relationships between contained datastreams, the media type of datastreams, etc. [5].

For the purpose of this paper two data elements added to DIDs during the ingestion process are of crucial importance:

- The DID-identifier: a globally unique identifier – a URI - for the DID itself. The DID-identifier should not be confused with the identifier of content contained in or referenced by a DID. Those identifiers are named Content-identifiers.
- The DID-creationTime: the time of creation of the DID, expressed as an ISO 8601 datetime [10] with seconds granularity.

3. STORING DIDS IN MULTIPLE OAI-PMH REPOSITORIES

Once a delivered asset has been turned into a DID by the ingestion process, the DID is stored in an OAI-PMH repository. Data assets at LANL are typically received in large batches, since secondary or primary publishers that account for the bulk of the data to be stored in the LANL Repository deliver weekly or annual feeds. In those cases, an autonomous OAI-PMH repository is created per delivered batch. As a result, many OAI-PMH repositories exist in the LANL Repository, each of which has the following characteristics:

- It has a unique, persistent `baseURL`, the http address `BaseURL(n)`
- Contained records are DIDs only
- The `identifier` used by the OAI-PMH is the DID-identifier
- The `datestamp` used by the OAI-PMH is the DID-creationTime.
- The only supported metadata format is DIDL, with `metadataPrefix` DIDL, defined by the MPEG-21 DIDL XML Schema. Because mapping a DID that represents a complex digital object to simple DC is quite an impossible task, support of DC by these OAI-PMH repositories is rather meaningless.
- The supported OAI-PMH harvesting `granularity` is at the seconds-level
- `Set` structures may be supported, but to reduce complexity this aspect will not be discussed in this paper.

As a result, each autonomous OAI-PMH repository can be harvested using a datestamp-based strategy, as a means to recurrently collect newly added DIDs from them.

Not only are new data assets at LANL typically received in batches, they are also quite stable in the sense that delivery of an update for an asset is rather rare. These ingestion properties have led to the creation of a special-purpose OAI-PMH repository, named the XMLtape. An XMLtape OAI-PMH repository is created as follows:

- As described earlier, when a batch of assets is delivered, each asset is turned into a DID.
- All those DIDs are concatenated into a single well-formed and valid XML file; this XML file can easily contain millions of DIDs.
- The XML file is then indexed using an approach inspired by a technique described by Google creators Page and Brin [11]: (1) the XML file is gzipped; (2) the gzipped file is indexed to support the core OAI-PMH keys, `identifier` and `datestamp`, which are respectively the DID-identifier and the DID-creationTime. The indexes record the values of these keys, and the byte-offset and byte-count in the gzipped file of the DIDs with a corresponding value.
- Software has been developed that makes the gzipped file and its corresponding index accessible through the OAI-PMH.

XMLtapes turn out to be a handy way to store large batches of stable assets. As the technique is based on the common, multi-platform gzip tool it provides guarantees for administration-less continuity. The nature of the indexes guarantees fast access to stored DIDs, and the simplicity of the components involved yields a high uptime of the XMLtape OAI-PMH repositories. Because of their XML format, XMLtapes can be validated using standard XML tools, and provide a high compression ratio. DIDs in XMLtapes are never updated. Rather, when an update for a contained asset is delivered, a new DID is created and stored in another OAI-PMH repository.

4. KEEPING TRACK OF AUTONOMOUS OAI-PMH REPOSITORIES: THE REPOSITORY INDEX

As has been shown, storing DIDs in a multitude of individual OAI-PMH repositories is attractive due to the nature of the ingestion properties at the LANL library. And, while updates can be harvested from each autonomous OAI-PMH repository, the question remains unanswered as to how harvesters operated by

downstream Service Providers - such as indexing engines - learn about the existence, addition of, and location of all those repositories. In order to provide this crucial intelligence, the Repository Index is introduced.

The Repository Index contains an entry for each autonomous OAI-PMH repository in the environment. Each entry contains the following information per OAI-PMH repository:

- The repository-baseURL: the `baseURL` of an OAI-PMH repository, which is a unique and persistent URI.
- The repository-creationTime: the time when the OAI-PMH repository becomes harvestable, by becoming visible through the Repository Index. This time is expressed as an ISO 8601 datetime with seconds granularity.
- Metadata pertaining to the creation of the OAI-PMH repository, its contents, etc.

It cannot be overlooked that the first two information elements of the Repository Index map directly to the notions of the `identifier` and the `datestamp` of the OAI-PMH, respectively. And, indeed, in the LANL Repository, the Repository Index is exposed as an OAI-PMH repository in its own right, with the following properties:

- It has a unique, persistent `baseURL`, the http address `BaseURL(Repo-Index)`.
- Contained records comply with a locally defined metadata format, identified by `metadataPrefix` INDEX, which facilitates the expression of the necessary metadata about autonomous OAI-PMH repositories.
- The identifier used by the OAI-PMH is the repository-baseURL, `BaseURL(n)`.
- The `datestamp` used by the OAI-PMH is the repository-creationTime. There are no updates to metadata contained in the Repository Index, and hence this `datestamp` will never change and always remain equal to the time the OAI-PMH repository became available for harvesting.
- The supported OAI-PMH harvesting `granularity` is seconds-level.
- `Set` structures may be supported, but to reduce complexity, this aspect will not be discussed in this paper. Typically, `set` structures would be used in the Repository Index to broadly categorize the nature or content of autonomous OAI-PMH repositories.

As a result of the introduction of the Repository Index, harvesters operated by downstream applications – all of which are internal to LANL – can use a datestamp-based harvesting strategy to gather newly added DIDs from the LANL Repository. Also the harvester contained in the OAI-PMH Federator, which is a special type of downstream application described later, will interact with the environment in the manner described here.

Presume that T1 and T2 are second-granularity datetimes, with T2 > T1, and that a harvester wants to collect DIDs added to the Repository since the last harvest, which was conducted at T1. These are the steps involved:

- The harvester issues a `ListIdentifiers` request against the Repository Index, using the `until` parameter with a value of T2. In response, the harvester receives a list of repository-baseURLs and associated repository-creationTimes.

```
[ BaseURL(Repo-Index)?
    verb=ListIdentifiers&until=T2&
    metadataPrefix=INDEX ]
```

- For each repository-baseURL that has a repository-creationTime larger than T1, the harvester issues a `ListRecords` request against the actual repository-baseURL. Since OAI-PMH repositories that meet this condition have become available for harvesting after the last harvest - all DIDs need to be collected from it – the harvester does not use the `from` argument in this request. It does, however, use the `until` parameter with a value of T2.

```
[ BaseURL(n)?
    verb=ListRecords&until=T2&metadataPrefix=DIDL]
```

- For each repository-baseURL that has a repository-creationTime smaller than or equal to T1, the harvester issues a `ListRecords` request against the actual repository-baseURL. Since these repositories were already available for harvesting at the time of the last harvest – only new or updated DIDs need to be collected – the harvester issues a `ListRecords` with a value of T1 for the `from` argument and a value of T2 for the `until` argument.

```
[BaseURL(m)?
    verb=ListRecords&from=T1&until=T2&
    metadataPrefix=DIDL ]
```

In order for these harvesting operations not to miss out on any updates or additions made to this highly distributed and dynamic environment, the following are crucial:

- Usage of the `until` parameter in the aforementioned harvesting requests.
- Synchronization of the clocks on all machines operating the autonomous OAI-PMH repositories and the Repository Index. This can be achieved using the Network Time Protocol [12].
- The content of the Repository Index must perfectly reflect the collection of harvestable OAI-PMH repositories in the environment. This tight synchronization of the Repository Index and the collection of harvestable repositories can be achieved in various ways. For example, the process of populating the Repository Index can be integrated with that part of the ingestion process where new OAI-PMH repositories are created, i.e. the ingestion process can write to the Repository Index. Alternatively, this can also be achieved using the OAI-PMH. For example, in the file-system based solution used by the LANL Repository, the OAI-PMH could be used as follows. When a new OAI-PMH repository is created, a special file containing the required information about that repository is written to the file system in which all OAI-PMH repositories reside. Using a tool similar to the 'OAI-PMH2 XML-file file-based OAI Data Provider' [13], this file system can be exposed as an OAI-PMH repository that has file names as `identifiers` and file-creation-dates – that become repository-creationTimes - as `datestamps`. Whenever the Repository Index receives the aforementioned `ListIdentifiers` request, it starts by

issuing a `ListRecords` request against this file-system-based OAI-PMH repository to collect all BaseURL(n) that are currently available. Next, the Repository Index is updated according to the response. Finally, being certain that its content matches the actual LANL Repository situation, the Repository Index can respond to the `ListIdentifiers` request it received from the downstream harvester. This OAI-PMH-based approach has not yet been tested in the LANL Repository effort.

5. A SPECIAL SERVICE PROVIDER: THE IDENTIFIER RESOLVER

As harvesters working on behalf of Service Providers collect DIDs from the LANL Repository, and as those Service Providers build services with the collected information, identifiers contained in the harvested DIDs become available in applications such as search engines. As mentioned before, these identifiers can either be DID-identifiers identifying the DIDs themselves, or Content-identifiers identifying content contained in DIDs. Obviously it is essential that, when such identifiers show up in downstream applications, the corresponding content can be retrieved from the Repository. For this purpose the Identifier Resolver is introduced to the environment. The Identifier Resolver is a special-purpose Service Provider that collects the information it requires from the Repository through recurrent OAI-PMH harvesting. From the harvested information, it only uses the DID-identifiers, the Content-identifiers, and the `baseURL` of the OAI-PMH repository in which these occur.

Table 1 illustrates the content of the Identifier Resolver. As can be seen, Id-1 and Id-4 are DID-identifiers, and the corresponding DIDs are located in the OAI-PMH repository with `baseURL` BaseURL(3) and BaseURL(6), respectively. As described in detail in [6] the Identifier Resolver also contains Content-identifiers as well as the location of the different versions of the corresponding content expressed as a combination of `baseURL` of an OAI-PMH repository and the DID-identifier of the DID in which the content resides. Since these Content-identifiers are not important for the OAI-PMH functionality of the LANL Repository described in this paper, they are not shown in Table 1.

Table1. Identifier Resolver Contents

DID-identifier	OAI-PMH repository
Id-1	BaseURL(3)
Id-4	BaseURL(6)

The Identifier Resolver is accessible to applications in a number of ways including the handle protocol [14], a SOAP-based mechanism, and a C library. After consultation of the Identifier Resolver, an application can use the OAI-PMH to retrieve the DID with a specified DID-identfier. For example, if the object with identifier Id-1 is requested, a look-up in the Identifier Resolver will learn that it is located at BaseURL(3). From this information, the application can conclude that the requested DID can be obtained by issuing the OAI-PMH request:

```
[ BaseURL(3)?
        verb=GetRecord&identifier=Id-1&
        metadataPrefix=DIDL ]
```

6. EXPOSING MULTIPLE AUTONOMOUS OAI-PMH REPOSITORIES AS A SINGLE ONE: THE OAI-PMH FEDERATOR

A new component – the OAI-PMH Federator - is introduced in the environment for the following reasons:

- As was described so far, harvesters – through the Repository Index – need to be aware of the location of each autonomous OAI-PMH repository in the environment in order to collect DIDs. This is not optimal, as those harvesters are not really interested in the autonomous repositories but rather in the new DIDs irrespective of their location.
- The infrastructure presented so far is only capable of disseminating DIDs as stored; the only supported `metadataPrefix` is DIDL. One can imagine that – for reasons of interoperability – the dissemination of stored DIDs rendered into other complex object representations such as METS [15], SCORM [16], IMS [17] would be desirable. And it would, for example, clearly be attractive if, in order to feed the Identifier Resolver, not the complete DIDs would have to be disseminated, but only their bare essentials as required by the Identifier Resolver. Rather than supporting these kinds of transformations at the level of each of the autonomous OAI-PMH repositories, a separate component, shared by all repositories, is introduced in the environment. This component, capable of disseminating and transforming DIDs, or content contained in DIDs, is named a Digital Item Processing engine and it operates according to the MPEG-21 Digital Item Processing (DIP) specification [18]. The functioning of the LANL DIP engine is described in detail in [6], and will, in this paper, be illustrated by means of a scenario.

The OAI-PMH Federator will relieve harvesters from the burden of having to interact with all autonomous OAI-PMH repositories, and having to understand about the Repository Index and the Identifier Resolver, by exposing the whole environment as a single OAI-PMH repository. As a matter of fact, the OAI-PMH Federator becomes the single point of access to the LANL Repository for harvesters, hiding the complexity of the LANL Repository environment from them.

The OAI-PMH Federator accepts incoming OAI-PMH requests from downstream harvesters, and contains logic to translate these requests into appropriate requests to be issued against the Repository Index, the Identifier Resolver and the autonomous OAI-PMH repositories. Since many of the latter requests are themselves OAI-PMH requests, the OAI-PMH Federator operates its private OAI-PMH harvester. Logic built in to the OAI-PMH Federator ensures that the responses received from the various components of the LANL Repository it interacts with are interpreted correctly, and, whenever appropriate, handed over to downstream harvesters as valid OAI-PMH responses.

The OAI-PMH Federator is an OAI-PMH repository with the following characteristics:

- It has a unique, persistent `baseURL`, the http address `BaseURL(Federator)`.
- The `identifier` used by the OAI-PMH is the DID-identifier.

- The `datestamp` used by the OAI-PMH is the DID-creationTime.
- DIDL is the natively supported metadata format, but, through dynamic processing of DIDs by the DIP engine, potentially many other metadata formats can be supported. The term metadata format must be interpreted broadly, as the `metadataPrefix` argument in harvesting requests issued against the OAI-PMH Federator can be used to express several types of transformations that can be applied to stored DIDs:
  - Transformations that map DIDL to another complex object model such as METS. In this case, the value for the `metadataPrefix` argument in harvesting requests could be METS, and the METS XML Schema would define the metadata format.
  - Transformations that filter information from stored DIDs, as is, for example, the case with harvesting of only identifiers by the Identifier Resolver. In this case, the metadata format will remain DIDL, but the nature of the harvesting request will need to be further clarified through the `metadataPrefix` , i.e. DIDL:identifiers.
- The supported `granularity` is seconds-level.
- In order to support harvesting from selected autonomous OAI-PMH repositories, if this would be required, the OAI-PMH Federator can expose an OAI-PMH `set` structure in which the `baseURL(n)` of each autonomous repository is presented as a `setSpec`.

The interaction of a downstream harvester with the LANL Repository through the OAI-PMH Federator is illustrated by detailing the manner in which the response to the following harvesting requests is provided: `ListMetadataFormats`, `ListSets`, `GetRecord`, `ListIdentifiers`.

6.1 ListMetadataFormats

```
[ BaseURL(Federator)?
      verb=ListMetadataFormats
```

and

```
 BaseURL(Federator)?
      verb=ListMetadataFormats&identifier=Id-1
```

where Id-1 is a DID-identifier]

Because both types of transformation described earlier can be applied to all DIDs, and because only DIDs are stored in the LANL Repository, support of a given metadata format is a Repository-wide property, that is not dependent on a specific DID or DID-identifier. Therefore, a response to the `ListMetadataFormats` verb – with or without `identifier` argument - is straightforward for the OAI-PMH Federator to create. The DIP engine holds a table – the DIP Table – that lists all methods that can be applied to objects stored in the LANL Repository depending on their nature, i.e. whether they are DIDs, whether they are assets, what the media type of an asset is, etc. The DIP Table has a section with multiple OAI-PMH-specific entries, each of which lists a `metadataPrefix` value, the associated XML Namespace, and a pointer to the method that can be used to transform a stored DID to the format identified by the `metadataPrefix` value. As a matter of fact, this section of the DIP Table stores all relevant information on all transformations that can be applied to DIDs, and this information directly corresponds to the metadata formats that are supported by the OAI-PMH Federator.

6.2 ListSets

```
[ BaseURL(Federator)?verb=ListSets  ]
```

The `ListSets` response detailing a `baseURL`-based `set` structure that reflects the `baseURLs` of autonomous OAI-PMH repositories, can easily be generated by the OAI-PMH Federator by issuing a `ListIdentifiers` request against the Repository Index

```
[ BaseURL(Repo-Index)?
     verb=ListIdentifiers&metadataPrefix=INDEX ]
```

and by transforming the response to that request into a `ListSets` response to the downstream harvester.

6.3 GetRecord

```
[ BaseURL(Federator)?
     verb=GetRecord&identifier=Id-1&
     metadataPrefix=DIDL
```

and

```
 BaseURL(Federator)?
     verb=GetRecord&identifier=Id-1&
     metadataPrefix=abc
```

in which Id-1 is a DID-identifier]

These are the steps involved in generating the appropriate `GetRecord` response:

- Via the DIP Table, the OAI-PMH Federator can determine whether the requested `metadataPrefix` - DIDL or abc – is supported. If yes, the process can continue, if not, a `cannotDisseminateFormat` error response can be generated.
- Through interaction with the Identifier Resolver, the OAI-PMH Federator finds out about the location of the DID with DID-identifier Id-1, namely `BaseURL(3).` If no entry for Id-1 would exist in the Identifier Resolver, the OAI-PMH Federator can generate an `idDoesNotExist` error response.
- The OAI-PMH Federator obtains the stored DID by issuing a `GetRecord` request

```
[    BaseURL(3)?
       verb=GetRecord&identifier=Id-1&
       metadataPrefix=DIDL ].
```

  - If the `metadataPrefix` requested in the original `GetRecord` request was DIDL, no special actions need to be undertaken.
  - If the `metadataPrefix` requested in the original `GetRecord` request was abc, the OAI-PMH Federator calls the DIP engine to have it apply the transform that – in the DIP Table – corresponds to abc.
- The OAI-PMH Federator embeds the record resulting from the previous step in a correct OAI-PMH response. If a `baseURL`-based `set` structure is exposed by the OAI-PMH Federator, this includes inserting `set` membership information in the `headers` of the responses .

6.4 ListIdentifiers

[BaseURL(Federator)?
verb=ListIdentifiers& from=T1&until=T2&
metadataPrefix=DIDL

and

BaseURL(Federator)?
verb=ListIdentifiers&from=T1&until=T2&
metadataPrefix=abc

in which T1 and T2 are as explained in Section 4]

These are the steps involved in generating the appropriate ListIdentifiers response:

- Via the DIP Table, the OAI-PMH Federator can determine whether the requested metadataPrefix - DIDL or abc – is supported. If yes, the process can continue, if not, a cannotDisseminateFormat error response can be generated.
- The OAI-PMH Federator conducts the different steps detailed in Section 4. Since support of a given metadata format is a Repository-wide attribute, the steps are identical for all supported metadataPrefix values:
  - Identification of the autonomous OAI-PMH repositories that meet the harvesting criteria by interaction with the Repository Index

    [BaseURL(Repo-Index)?
    verb=ListIdentifiers&until=T2&
    metadataPrefix=INDEX]

  - For each OAI-PMH repository identified through interaction with the Repository Index one of the following actions is taken: (1) If the OAI-PMH repository was created after the last harvest, collect all DID-identifiers

    [BaseURL(n)?
    verb=ListIdentifiers&until=T2&
    metadataPrefix=DIDL];

    (2) If the OAI-PMH repository already existed at the last harvest, collect the DID-identifiers of added or updated DIDs

    [BaseURL(n)?
    verb=ListIdentifiers&from=T1&until=T2&
    metadataPrefix=DIDL]

- The OAI-PMH Federator returns the responses to harvesting requests issued against the individual OAI-PMH repositories as valid OAI-PMH responses to the downstream harvester. As was the case with the GetRecord response, this might include editing the headers of the responses to insert set-membership information. The following are important with respect to this step in the process:
  - It is – in theory – possible that the Repository Index returns a noRecordsMatch error response. The OAI-PMH Federator must return such a response to the downstream harvester.
  - It is possible that autonomous OAI-PMH repositories respond with a noRecordsMatch error response. The OAI-PMH Federator must not pass on such responses to the downstream harvester but rather interpret them as a command to start harvesting from the next autonomous OAI-PMH repository that was returned by the Repository Index. Only if no meaningful responses have been received from any of the individual OAI-PMH repositories must the OAI-PMH Federator itself return a noRecordsMatch error response.
  - Care must be taken to appropriately handle resumptionTokens delivered by an individual OAI-PMH repository. As a matter of fact, the OAI-PMH Federator will need to adapt such resumptionToken by adding the following information to it (1) the baseURL of the autonomous OAI-PMH repository from which the resumptionToken was received (2) the requested metadataPrefix. Also, the OAI-PMH Federator may need to create resumptionTokens of its own, to make the transition between harvesting from one autonomous OAI-PMH repository to the next easier.
  - If one of the autonomous OAI-PMH repositories to be harvested from returns a badResumptionToken error message, the OAI-PMH Federator must pass this on to the downstream harvester. If one of the OAI-PMH repositories fails to respond, the OAI-PMH Federator must generate an appropriate HTTP error indicating this 'internal error'. The HTTP status-code 503 'service unavailable' is suitable for that purpose. In both cases, the responses indicate to the downstream harvester that its ongoing harvest can not be completed successfully, and that the intended harvest should be restarted at some later time.

A ListIdentifiers request with a set argument that specifies the baseURL of an autonomous OAI-PMH repository can be obtained by first interacting with the Repository Index – using a GetRecord request - to determine whether the specified repository exists. Then, the harvesting request targeted at the OAI-PMH Federator can be translated to a request targeted at an individual OAI-PMH repository by using the value of the set argument of the initial request as the baseURL for the translated request. The exact harvesting request to be issued will depend on the relationship between the time of the last harvest and the creation time of the OAI-PMH repository to be harvested from, as was explained earlier in this Section and in Section 4. For example, if repository-creationTime(BaseURL(6)) > T1 then the OAI-PMH Federator can translate the incoming request

[BaseURL(Federator)?
verb=ListIdentifiers&from=T1&until=T2&
metadataPrefix=DIDL&set=BaseURL(6)]

to

[BaseURL(6)?
verb=ListIdentifiers&until=T2&
metadataPrefix=DIDL].

In this case, also set information needs to be added to a resumptionToken when it is passed on to a downstream harvester.

Through a process similar to the above, the response to a `ListIdentifiers` request without a `from` and/or `until` argument can be generated.

The process of responding to a `ListRecords` request is similar to that of responding to a `ListIdentifiers` request. However, as was the case with the `GetRecord` request, when a metadata format other than DIDL is requested, the DIP engine will be called for each DID received in the response from autonomous OAI-PMH repositories. The `ListRecords` response delivered by the OAI-PMH Federator will contain the requested transformation of the stored DIDs.

7. DISCUSSION

Initial versions of each of the components of the described architecture have been implemented, and a series of small-scale experiments in which DIDs – or transformations thereof - were harvested from the environment through the OAI-PMH Federator were completed successfully. At the time of writing, a larger scale test in which millions of DIDs will be ingested into multiple autonomous OAI-PMH repositories is being prepared, and a transition of the described architecture into production is expected to happen in the next few months. By making the LANL Repository harvestable through the OAI-PMH Federator, each downstream application can use off-the-shelf OAI-PMH harvesting software to collect added or updated materials. The OAI-PMH Federator itself can be built around off-the-shelf OAI-PMH harvesting software; the OCLC OAICat framework [19] has been selected for that purpose. Also, off-the-shelf OAI-PMH repository software can be used to deploy both the Repository Index, and the multitude of autonomous OAI-PMH repositories. Generally speaking, the multi-faceted use of the OAI-PMH in the LANL Repository ensures that only a very limited set of lightweight tools, most of which are available off-the-shelf, are required to interact with the environment.

To reduce the complexity of the paper, the use of OAI-PMH `sets` by the autonomous OAI-PMH repositories or the Repository Index, and the impact thereof on the OAI-PMH Federator, has not been discussed. However, their use is currently being actively researched. Hence the question is not whether `sets` can be supported at all, because the answer to that question is affirmative. The question is rather for which purpose the `set` structure will be used, and how exactly it will be implemented. Since `sets` are a technique provided by the OAI-PMH to allow for selective harvesting, it can be used to achieve various optimizations in the proposed solution aimed at guaranteeing its scalability. As was mentioned, XMLtapes are static and hence never need to be harvested for updates after an initial harvest gathered all contained DIDs. As such, many polls of autonomous OAI-PMH repositories by the OAI-PMH Federator could be avoided if the Repository Index would maintain a `set` structure that reflects the static or dynamic nature of those repositories, and if the OAI-PMH Federator would make use of this `set`-information into in the logic that underlies its interaction with the environment. Other optimizations can be imagined that would allow downstream harvesters to only collect the type of DIDs they are really interested in, rather than to have them collect all DIDs and have them dispose of the ones that are not relevant to their task. For example, a full-text indexing engine is really only interested in DIDs containing textual materials, while a video indexing engine only wants those that contain moving images. This would suggest an optimization that could be achieved by implementing a `set` structure supported throughout the LANL Repository reflecting media types. This would also be attractive for the purpose of general repository management and digital preservation. Another potential `set` structure could be collection-oriented, and would allow a downstream harvester to, for example, only collect DIDs that contain Inspec data. Generally speaking, it seems that the nature of the `set` structure to be supported is closely related with the specific requirements of the anticipated applications on whose behalf the downstream harvesters operate. It also seems that `sets` supported by all autonomous OAI-PMH repositories in the environment are of particular interest for the purpose of optimizing harvests.

The simplicity of the tools used in the architecture, and their off-the-shelf availability may make the proposed solution attractive for institutions beyond LANL that share the need to store collections of complex digital objects. As was described, care must be taken of the appropriate time-synchronization of the autonomous OAI-PMH repositories, the Repository Index and the OAI-PMH Gateway in order to ensure that downstream harvesters do not miss out on updates in the environment. Also, a high uptime of the OAI-PMH repositories must be guaranteed in order to avoid unsuccessful harvests. These boundary conditions of the solution can straightforwardly be met in the controlled environment of the repository of a single institution. Of special interest to certain institutions may be the XMLtape, which is particularly useful for archiving a collection of static objects in a self-contained manner, and to make those objects accessible using tools supported on most operating systems, and a protocol – the OAI-PMH – that is straightforward to implement. As this yields hassle-free operation and high uptimes – properties appreciated by many - LANL has decided to make its XMLtape software available to the public [20].

The protocol-based nature of the proposed solution, and its modularity, may also make it attractive for the federation of groups of OAI-PMH repositories distributed over the Web. For example, such a federation could consist of Institutional Repositories that operate various brands of Institutional Repository software, use different document models, and yet have the need to recurrently interchange contained objects. Also, another federation can be imagined that groups trusted digital repositories [21] that regularly need to interchange objects in order to guarantee redundancy. Generally speaking, a federation of community-based OAI-PMH repositories containing complex digital objects can be imagined for which a Repository Index and an OAI-PMH Federator are deployed. Records contained in those repositories could be homogeneous across the federation with each repository supporting, for example, DIDL. But the federation could also be heterogeneous, with one repository supporting DIDL, another METS, and yet another IMS. The population of a Repository Index in this scenario would be different than the one described in the context of the LANL Repository but its OAI-PMH functionality would remain identical. Again, an OAI-PMH Federator hides the complexity of the multitude of repositories to harvesters, and through a component similar to the DIP engine, it could support crosswalks between the multiple complex object format(s) used by the federation. Assuming that a crosswalk between each of those formats is available, then the combination of a `ListMetadataFormats` request against a repository in the

federation and a lookup of the resulting `metadataPrefix` in the equivalent of the DIP Table suffice for the OAI-PMH Federator to be able to respond to a harvest requesting objects in whichever format supported in the federation. As a result, the OAI-PMH Federator would operate as a 'complex object format' switch for the federation. In such a distributed Web-based federation, the the synchronicity requirement can be met using available network tools, as was described for the LANL Repository. This would obviously require an appropriate level of organization of the federation. The same is true for guaranteeing a high uptime of the OAI-PMH repositories in the federation.

The question of more general applicability of the proposed architecture becomes harder to answer when loosely-structured or unmanaged federations are considered. Consider, for example, the collection of all public OAI-PMH repositories as a federation. This is not a federation of OAI-PMH repositories that expose complex objects, but rather one in which more regular metadata formats – such as DC and MARCXML – are supported. This does not really influence the nature of the architecture. In this federation, an interesting parallel can be drawn between the Repository Index and the registries operated by the OAI [22] and UUIC [23], as the latter list the `baseURLs` of all repositories in this loose federation. Also, the ERRoLs service [24] capable of resolving oai-identifiers [25] in a sense resembles the Identifier Resolver, although it uses business rules rather than data collected from individual repositories to resolve identifiers. This suggests that an OAI-PMH Federator might potentially be added to this federation as a single point of access to all public OAI-PMH repositories. However, since neither the synchronicity requirement nor the high uptime of repositories seems straightforward to implement in such a loosely-structured federation, further research would be required to determine the usability of the proposed solution in that realm. Clearly, such a loosely-structured federation would require a flexible variation of the proposed OAI-PMH Federator that relies on a schedule-based scheme to achieve a form of pseudo-synchronicity. Meanwhile, the aggregating approach, as proposed by Celestial [26], seems more appropriate for such loosely-structured federations. Celestial, an OAI-PMH aggregator, actually collects the records contained in all repositories, stores them in its own environment, and re-exposes them at a single `baseURL`. In contrast, the OAI-PMH Federator merely acts as a gateway to the repositories, collecting records from those repositories and immediately passing them on to downstream harvesters, as such avoiding the central storage space required by Celestial.

To be complete, it should be mentioned that, due to their lack of support of DC, neither of the OAI-PMH repositories used in the proposed architecture are compliant with the current version of the OAI-PMH that mandates support of DC. As the described use of the OAI-PMH seems appropriate, suitable and attractive for the described problem domain, this seems to further fuel the discussion [27] as to whether DC should indeed remain mandatory.

8. CONCLUSION

This paper has focused on the multi-faceted use of the OAI-PMH in the LANL repository architecture. Official and/or de-facto standards are used throughout this architecture to store and make accessible a vast collection of scholarly asset in a consistent and sustainable way. These include MPEG-21 DIDL, MPEG-21 DIP, NISO OpenURL, and the OAI-PMH. Other papers by the DL Research and Prototyping Team are available that provide details on the use of MPEG-21 DIDL as the format to represent complex digital objects, and on the application of NISO OpenURL and MPEG-21 DIP to request disseminations of selected objects from the LANL Repository.

In essence, this paper has described an approach to uniformly make a vast and, ever growing data collection available to various downstream applications. Each of these applications focuses on building accurate services on top of the data collection and therefore must be able to remain permanently in sync with it. In the proposed approach, this rather complex problem is modularized through the introduction of several interacting components - the individual OAI-PMH repositories, the Repository Index, the Identifier Resolver and the OAI-PMH Federator - each addressing a simpler sub-problem. For most of the interactions between the components, the lightweight OAI-PMH protocol plays a prominent role. This makes the proposed approach attractive, as the OAI-PMH is a lightweight protocol for which software tools are readily available. It may also make the approach attractive beyond the LANL Research Library. A more general applicability seems to be feasible, indeed, for OAI-PMH repositories under control of a single institution, or for a well-managed federation of OAI-PMH repositories. Further research would be required to determine whether and how the solution could be adapted to enable its deployment in loosely-structured federations.

9. ACKNOWLEDGMENTS

The authors would like to thank their colleagues Luda Balakireva, Jeroen Bekaert and Thorsten Schwander from the LANL DL Research and Prototyping team for their contributions to the reported work, and the LANL library director, Rick Luce, for his ongoing support.

10. REFERENCES

[1] J. Bollen, R. Luce. Evaluation of Digital Library Impact and User Communities by Analysis of Usage Patterns. D-Lib Magazine June 2002 Volume 8 Number 6. http://dx.doi.org/10.1045/june2002-bollen

[2] H. Van de Sompel, J. Young, and T. Hickey. Using the OAI-PMH ... Differently. D-Lib Magazine, July/August 2003, Volume 9 Number 7/8. http://dx.doi.org/10.1045/july2003-young

[3] M. Smith et al. DSpace An Open Source Dynamic Digital Repository. D-Lib Magazine.January 2003, Volume 9 Number 1. http://dx.doi.org/10.1045/january2003-smith

[4] T. Staples et al.The Fedora Project An Open-source Digital Object Repository Management System. D-Lib Magazine April 2003 Volume 9 Number 4. http://dx.doi.org/10.1045/april2003-staples

[5] J. Bekaert, P. Hochstenbach, and H. Van de Sompel. Using MPEG-21 DIDL to Represent Complex Digital Objects in the Los Alamos National Laboratory Digital Library. D-Lib Magazine, November 2003, Volume 9 Number 11. http://dx.doi.org/10.1045/november2003-bekaert

[6] J. Bekaert, L. Balakireva , P. Hochstenbach, and Herbert Van de Sompel. Using MPEG-21 DIP and NISO OpenURL for

the dynamic dissemination of Complex Digital Objects in the Los Alamos National Laboratory Digital Library. D-Lib Magazine, February 2004, Volume 10 Number 2. http://dx.doi.org/10.1045/february2004-bekaert

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

[8] NISO committee AX. ANSI/NISO Z39.88-2004. The OpenURL Framework for Context-Sensitive Services. November 2003 http://library.caltech.edu/openurl/StandardDocuments/Part1-Ballot-20031111.pdf

[9] MPEG-21, Information Technology, Multimedia Framework, Part 2: Digital Item Declaration, ISO/IEC 21000-2:2003, March 2003.

[10] ISO 8601:2000, Data elements and interchange formats -- Information interchange -- Representation of dates and times, Technical committee TC 154, ICS 01.140.30,stage 60.60, 2000-12-21

[11] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In Proceedings of the 7th International World Wide Web Conference, Brisbane, Australia, 1998. http://citeseer.nj.nec.com/brin98anatomy.html

[12] D. Mills. Network Time Protocol RFC 2030. http://www.eecis.udel.edu/~mills/database/rfc/rfc2030.txt

[13] Hussein Suleman.OAI-PMH2 XMLFile File-based Data Provider. December 2002, http://www.dlib.vt.edu/projects/OAI/software/xmlfile/xmlfile.html

[14] S. Sun, et al. Handle System Overview. Internet Engineering Task Force (IETF) Request for Comments (RFC), RFC 3650, November 2003. http://hdl.handle.net/4263537/4069

[15] Metadata Encoding and Transmission Standard (METS), http://www.loc.gov/standards/mets/

[16] The Sharable Content Object Reference Model (SCORM). http://www.adlnet.org/index.cfm?fuseaction=scormabt

[17] IMS Global Learning Consortium, “IMS Content Packaging XML Binding - version 1.1.2 - Final specification,” 2001.

[18] MPEG-21, Information Technology, Multimedia Framework, Part 10: MPEG-21 Digital Item Processing, ISO/IEC JTC1/SC29/WG11 N5855, Trondheim, July 2003.

[19] OAICat. http://www.oclc.org/research/software/oai/cat.htm

[20] Yet Another Repository (YAR). http://yar.sourceforge.net

[21] RLG-OCLC. Trusted Digital Repositories: Attributes and Responsibilities. http://www.rlg.org/longterm/repositories.pdf

[22] Open Archives Registry, http://www.openarchives.org/Register/BrowseSites.pl

[23] Experimental OAI Registry at The University of Illinois at Urbana Champaign http://gita.grainger.uiuc.edu/registry

[24] ERRoLs. http://www.oclc.org/research/projects/oairesolver/

[25] Van de Sompel, H., Lagoze, C., Nelson, M., and Warner, S. Implementation Guidelines for the Open Archives Initiative for Metadata Harvesting: Specification and XML Schema for the OAI Identifier Format, 2002 http://www.openarchives.org/OAI/2.0/guidelines-oai-identifier.htm

[26] Tim Brody et al. Digitometric Services for Open Archives Environments. Proceedings of European Conference on Digital Libraries 2003, pages pp. 207-220, Trondheim, Norway.

[27] OAI-implementers mailing list, thread “Reconsidering mandatory DC in OAI-PMH”. http://www.openarchives.org/pipermail/oai-implementers/2003-August/000945.html