Network Information System

NESCent Dryad Subcontract (Year 1)

Metacat OAI-PMH – Project Plan 25 February 2009

Mark Servilla
servilla@lternet.edu

LTER Network Office
Department of Biology, MSC03 2020
1 University of New Mexico
Albuquerque, NM 87131-0001

Revision History

Date	Version	Description	Author
25 Feb 2009	1.0	Original document creation.	M. Servilla
2 March 2009	1.0	Added R. Scherle's comments to section 3.2.2.	M. Servilla
18 March 2009	1.0	Added EML to DC crosswalk mappings table to section 5.2.1.1.	D. Costa
27 March 2009	1.0	Completed first pass at the 6 OAI-PMH verb use cases.	M. Servilla
11 August 2009	1.0	Update at conclusion of Year 1 work.	D. Costa
13 August 2009	1.0	Final edits.	M. Servilla

Table of Contents

1 Introduction....5
2 Background....5
2.1 Dryad Project....5
2.2 OAI-PMH....5
2.3 Metacat....6
2.4 Ecological Metadata Language....7
3 Scope of Work....7
3.1 Metadata Cross-walks....7
3.1.1 EML and DC....7
3.1.2 EML and Dryad Application Profile....8
3.2 OAI-PMH Metacat Service Interfaces....8
3.2.1 Repository....8
3.2.2 Harvester....8
4 Use Case Scenarios....9
4.1 Repository....9
4.1.1 GetRecord....9
4.1.2 Identify....10
4.1.3 ListIdentifiers....10
4.1.4 ListMetadataFormats....11
4.1.5 ListRecords....12
4.1.6 ListSets....13
4.2 Harvester....13
5 Development Plan....14
5.1 Metacat SVN Integration....14
5.2 Design Details....15
5.2.1 Crosswalks....15
5.2.1.1 EML to DC crosswalk....15
5.2.2 Repository....16
5.2.3 Harvester....17
6 Schedule....19
6.1 Cross-Walks....19
6.2 Repository....19
6.3 Harvester....19
7 Generalized Definitions....20
7.1 OAI-PMH....20
8 Informational Links....21

9 OAI-PMH Error Codes.....22

1 Introduction

The following document provides a road map for the LTER Network Office sub-contract with the NESCent Dryad project. In particular, this project planning document addresses work to be performed in Year 1 – the design and development of a Metacat Open Archives Initiative – Protocol for Metadata Harvesting (OAI-PMH) service interface that includes adapters for Metacat acting as both an OAI-PHM "repository" and "harvester". Dependent work includes the design and development of metadata cross-walk between the Ecological Metadata Language (EML), Dublin Core (DC), and the Dryad Application Profile (DAP; an extended version of DC). A final procedural goal of this work will entail the bi-directional transfer of metadata between the LTER Data Catalog Metacat server (EML-based metadata only) and the NESCent Dryad server.

2 Background

2.1 Dryad Project

The Dryad project is a funded National Science Foundation grant under the Directorate for Biological Sciences' Biological Databases and Informatics program. The goal of the Dryad project is to develop a metadata and data repository that supports the discovery and management of data associated with publications in the field of evolutionary biology. In addition, the Dryad repository will act as a clearing-house for data/metadata from related fields in biological and environmental sciences. To achieve data/metadata interoperability, the Dryad will adopt the OAI-PMH standard and become both an OAI-PMH "repository" and "harvester", thereby exchanging metadata (and references to data) with other OAI-PMH compliant repositories. A sub-goal of the Dryad project is to perform metadata exchange with the Metacat XML database that underlies the LTER Data Catalog. As such, this particular goal is the focus of this document, and requires the design and development of an OAI-PMH service interface for Metacat. The Dryad project (and Metacat) will also support implementation of the Library of Congress Search and Retrieve via URL (SRU) standard, which will allow on-the-fly access to repository contents by third parties through a web-service protocol, and will also enable syndication of repository contents (the SRU work is scheduled for year 2 of the NESCent/Dryad LTER subcontract).

2.2 OAI-PMH

The Open Archives Initiative – Protocol for Metadata Harvesting was first developed in the late 1990's as a standard for harvesting metadata from distributed metadata/data repositories. The current version of the OAI-PMH standard is 2.0 as of June 2002, with minor updates in December 2008 (http://www.openarchives.org/OAI/openarchivesprotocol.html).

The OAI-PMH standard uses the Hyper-text Transport Protocol (HTTP) as a transport layer and specifies six query methods (called "verbs") that must be supported by an OAI-PMH compliant "repository" (see Section 7.1). These methods are:

1. **GetRecord** – retrieves zero or one complete metadata record from a repository;
2. **Identify** – retrieves information about a repository;

3. **ListIdentifiers** – retrieves zero or more metadata record “headers” (not the complete metadata record) from a repository;
4. **ListMetadataFormats** – retrieves a list of available metadata record formats supported by a repository;
5. **ListRecords** – retrieves zero or more complete metadata records from a respository; and
6. **ListSets** – retrieves the set structure from a repository.

The OAI-PMH compliant repository must accept requests in both HTTP-GET and HTTP-POST formats. Responses from the repository must be returned as an XML-encoded (version 1.0) stream. Error handling must be supported by the repository and provide the correct error response code back to the harvester. Detailed specifications and examples of all six methods may be viewed in Section 4 of the standards document (http://www.openarchives.org/OAI/openarchivesprotocol.html# ProtocolMessages).

2.3 Metacat

The Metacat is a schema independent XML database that utilizes a relational table framework to store decomposed content of XML documents, while still supporting path-based queries. The Metacat project is part of the larger Knowledge Network for Biocomplexity (KNB) grant (DEB99–80154) that was funded in 1999 by the National Science Foundation and under the Division of Environmental Biology program. The Metacat server itself is a set of Java servlets designed to interface with the Metacat database, in addition to providing support for both uni- and bi-directional replication between Metacat servers, user authentication (via LDAP bindings), Life Science Identifiers (LSIDs), map-based geographic services, and other feature-rich adapters (this work will perhaps become such an “adapter”). Metacat is capable of storing metadata and data; data can be in both text-based and binary formats. There are a number of client interfaces to the Metacat server, including those written in Java, Perl, Python, Ruby, and PHP.

The Metacat servlet API supports inserting, querying, and retrieving metadata/data. There is also a separate Metacat harvester that can automate timed harvests of metadata/data from remote sites. The harvester relies on a digest encoded in XML that points to individual metadata documents at the remote site. The harvester processes this digest on a schedule defined by the remote site and attempts to insert each document into the “home” instance of Metacat; an insertion occurs only if the document is valid XML, verified against an XML schema if one exists, and if it is not already within the database. New versions of a metadata document never overwrite older versions, thereby preserving the lineage history of a particular document.

Queries for metadata are performed through the search API and are composed as a Metacat specific expression encoded as XML, which is eventually translated to a standard SQL expression for the underlying database query. Query results are returned as an XML list of documents that satisfy the query expression. Individual metadata documents may be retrieved by a standard URL or as an LSID identifier. Metacat can return a metadata document as native XML or as HTML in one of many presentation formats called “skins”.

The Metacat project is ongoing and is anticipating the release of Metacat 1.9 in March 2009. The project itself is considered open source and under GPL licensing. More information on Metacat, including extensive documentation, can be obtained through the ecoinformatics.org website (http://knb.ecoinformatics.org/software/metacat/).

2.4 Ecological Metadata Language

The Ecological Metadata Language (EML), also funded as part of the KNB grant, is an XML schema specification for describing scientific objects, including datasets, bibliographic citations, software, and methodologies. The EML standard has been adopted by many organizations in the ecological community; the Long Term Ecological Research (LTER) Network officially adopted the EML 2.0 specification as their metadata standard in May 2003. Although a new standard, EML has incorporated content structure from other widely used metadata standards, including Dublin Core and the Federal Geographic Data Committee's Content Standard for Digital Geospatial Metadata. The EML is a fine-grained structure composed of many optional elements. As such, EML-encoded metadata can generally be translated to other coarser-grained standards, but the reverse is not necessarily true.

Metadata can be entered into the EML through various clients, such as Morpho (http://knb.ecoinformatics.org/software/morpho/) and other organic applications that have been developed at individual research sites. Once generated, most EML finds its way into either local repositories or an instance of a Metacat database. For the LTER Data Catalog Metacat, a majority of metadata documents are of the EML standard.

EML 2.0.1 is the current standard specification, however EML 2.1 is due to be released in March 2009. Additional information about EML and the source schema may be found at the ecoinformatics.org website (http://knb.ecoinformatics.org/ software/ eml/).

3 Scope of Work

The Dryad project grant includes a subcontract award to the LTER Network Office to design and develop three specific products: (1) metadata cross-walks between the EML and DC and the EML and Dryad Application Profile, (2) an OAI-PMH compliant Metacat service interface to act as both a “repository” and “harvester”, and (3) a Metacat service interface that supports the Library of Congress Search and Retrieve via URL (SRU) standard (the Search and Retrieve by Web-service SRW is a special case of the SRU standard and will also be supported) . Products 1 and 2 will be addressed in Year 1 (this work), while product 3 will be the focus of Year 2 (not addressed in this document).

3.1 Metadata Cross-walks

3.1.1 EML and DC

Effort will include identifying the mapping of content elements between the EML and DC. The EML (described above) is a fine-grained structure that has many content elements in common with DC. For transformations from the EML to DC, only “simple” or “unqualified” Dublin Core (using only the 15 canonical elements) will be the target of

the transformation – OAI-PMH requires unqualified Dublin Core metadata be supported as a minimum. The cross-walk mapping will be approved by the primary contractor prior to development of the XSLT script. Once the mapping elements are identified, an Extensible Stylesheet Language Transformation (XSLT) script will be written for performing the actual transformation. The XSLT will be tested for accuracy and completeness.

3.1.2 EML and Dryad Application Profile

Effort will include identifying the mapping of content elements between the EML and Dryad Application Profile (DAP). The DAP includes content elements from DC, the Data Documentation Initiative, Darwin Core, EML, and PREMIS. The cross-walk mapping will be approved by the primary contractor prior to development of the XSLT script. Once the mapping elements are identified, an Extensible Stylesheet Language Transformation (XSLT) script will be written for performing the actual transformation. The XSLT will be tested for accuracy and completeness.

Discussions with Ryan Scherle and Hilmar Lapp on 26 May 2009 resolved that the OAI-PMH Metacat "Provider" service will not be required to support the Dryad Application Profile standard since a viable crosswalk between EML and DAP is not possible due to required elements in DAP that are not consistently available in EML. Conversion of EML to the Dryad Application Profile will be addressed at a later date by the Dryad development team.

3.2 OAI-PMH Metacat Service Interfaces

Effort will include designing and implementing OAI-PMH Metacat "repository" and "harvester" service interfaces.

3.2.1 Repository

The design and implementation of the OAI-PHM Metacat Repository service interface will make available all six OAI-PMH methods (GetRecord, Identify, ListIdentifiers, ListMetadataFormats, ListRecords, and ListSets) as defined in the OAI-PMH Version 2 Specification (http://www.openarchives.org/OAI/openarchivesprotocol.html) through a standard HTTP URL that accepts both HTTP-GET and HTTP-POST formats. The design will attempt to be metadata neutral with respect to other metadata standards residing within a Metacat instance, however the implementation will only address EML metadata. The design will utilize example OAI-PMH repositories (e.g., OCLC's OAIcat, UIUC provider, or the DLESE provider) as guidance.

3.2.2 Harvester

The design and implementation of the OAI-PMH Metacat Harvester service interface will utilize all six OAI-PMH methods to request metadata or related information from another OAI-PMH compliant repository using a standard HTTP URL in either an HTTP-GET or HTTP-POST format and transform such metadata into the EML standard, which will subsequently be inserted into a Metacat instance. The design will initially harvest DAP data, allowing fall-back to Dublin Core when DAP is not available. The design will

allow harvesting of other metadata types in the future. The design will utilize example OAI-PMH harvesters (e.g., OCLC's OAIHarvester2, UIUC harvester, or the DLESE harvester) as guidance.

4 Use Case Scenarios

4.1 Repository

The following use case scenarios describe the high-level interactions between a remote OAI-PMH harvester and the local Metacat OAI-PMH Repository (MOR). In all use cases, the 'harvester' is represented by a remote client, while the 'repository' is represented by the MOR service.

4.1.1 GetRecord

Characteristics Information:

Goal in Context:	Returns an individual metadata record from the repository to the requesting harvester.
Scope:	
Level:	
Pre-Condition:	MOR service is listening for HTTP URL requests.
Success End Condition:	The metadata record is returned to the requesting harvester.
Failed End Condition:	The metadata record is not returned and/or a malformed metadata record is returned to the requesting harvester.
Primary Actor:	MOR service.
Trigger Event:	The requesting harvester issues an HTTP URL 'GetRecord' verb request.

Main Success Scenario:

Step	Actor	Action Description
1.	Harvester	Issues an HTTP URL 'GetRecord' request.
2.	MOR service	Accepts and parses request.
3.	MOR service	Identifies 'verb' and validates URL string for correct arguments; confirms validity of metadata format type 'metadataPrefix'.
4.	MOR service	Requests metadata record document type from Metacat server.
5.	MOR service	Confirms availability of export metadata format identified by the 'metadataPrefix' arugment and the metadata record document type.
6.	MOR service	Requests metadata document from Metacat server identified by the 'identifier' argument (document ID).
7.	MOR service	Performs metadata translation to export metadata format.
8.	MOR service	Composes OAI-PHM record.
9.	MOR service	Returns OAI-PMH record to requesting harvester as an XML-encoded byte stream.

Scenario Extensions: None

Step	Condition	Action Description

Scenario Variations

Step	Actor	Action Description

4.1.2 Identify

Characteristics Information:

Goal in Context:	Returns information about the repository to the requesting harvester.
Scope:	
Level:	
Pre-Condition:	MOR service is listening for HTTP URL requests.
Success End Condition:	Repository information is returned to the requesting harvester.
Failed End Condition:	Repository information is not returned or incorrect information is returned to the requesting harvester.
Primary Actor:	MOR service.
Trigger Event:	The requesting harvester issues an HTTP URL 'Identify' verb request.

Main Success Scenario:

Step	Actor	Action Description
1.	Harvester	The requesting harvester issues an HTTP URL 'Identify' request.
2.	MOR service	Accepts and parses request.
3.	MOR service	Identifies 'verb' and verifies URL string does not contain additional arguments.
4.	MOR service	Requests metadata time-stamp boundaries from Metacat.
5.	MOR service	Composes 'Identify' response package.
6	MOR service	Returns response package to requesting harvester as an XML-encoded byte stream.

Scenario Extensions: None

Step	Condition	Action Description

Scenario Variations

Step	Actor	Action Description

4.1.3 ListIdentifiers

Characteristics Information:

Goal in Context:	Returns 0 or more record 'headers' from the repository to the requesting harvester.
Scope:	
Level:	
Pre-Condition:	MOR service is listening for HTTP URL requests.
Success End Condition:	The correct record 'headers' are returned to the requesting harvester.
Failed End Condition:	Record 'headers' are not returned or the incorrect record 'headers' are returned to the requesting harvester.
Primary Actor:	MOR service

Trigger Event:	The requesting harvester issues an HTTP URL 'ListIdentifiers' verb request.

Main Success Scenario:

Step	Actor	Action Description
1.	Harvester	The requesting harvester issues an HTTP URL 'ListIdentifiers' verb request.
2.	MOR service	Accepts and parses request.
3.	MOR service	Identifies 'verb' and validates URL string for correct arguments; confirms validity of metadata format type 'metadataPrefix'.
4.	MOR service	Retrieves all metadata document identifiers from the Metacat server.
5.	MOR service	For each document identifier, request metadata record document type from Metacat server.
6.	MOR service	For each document identifier and document type that meets the metadata format ('metadataPrefix') criteria, add to ListIdentifier 'list'.
7.	MOR service	Return 'list' to requesting harvester as an XML-encoded byte stream.

Scenario Extensions: None

Step	Condition	Action Description

Scenario Variations

Step	Actor	Action Description

4.1.4 ListMetadataFormats

Characteristics Information:

Goal in Context:	Returns a list of metadata formats supported by the MOR service to the requesting harvester.
Scope:	
Level:	
Pre-Condition:	MOR service is listening for HTTP URL requests.
Success End Condition:	A list of metadata formats is returned to the requesting harvester.
Failed End Condition:	A list of metadata formats is not returned or a incorrect list of metadata formats is returned to the requesting harvester.
Primary Actor:	MOR service
Trigger Event:	The requesting harvester issues an HTTP URL 'ListMetadatFormats' verb request.

Main Success Scenario:

Step	Actor	Action Description
1.	Harvester	The requesting harvester issues an HTTP URL 'ListMetadatFormats' verb request.
2.	MOR service	Accepts and parses request.
3.	MOR service	Identifies 'verb' and validates URL string for correct arguments.
4.	MOR service	Composes a 'list' of supported metadata formats.
5.	MOR service	Returns 'list' to requesting harvester as an XML-encode byte stream.

Scenario Extensions: None

Step	Condition	Action Description

Scenario Variations

Step	Actor	Action Description

4.1.5 ListRecords

Characteristics Information:

Goal in Context:	Returns 0 or more metadata records from the repository to the requesting harvester.
Scope:	
Level:	
Pre-Condition:	MOR service is listening for HTTP URL requests.
Success End Condition:	The correct records are returned to the requesting harvester.
Failed End Condition:	The incorrect number of records or incorrect records are returned to the requesting harvester.
Primary Actor:	MOR service
Trigger Event:	The requesting harvester issues an HTTP URL 'ListRecords' verb request.

Main Success Scenario:

Step	Actor	Action Description
1.	Harvester	The requesting harvester issues an HTTP URL 'ListRecords' verb request.
2.	MOR service	Accepts and parses request.
3.	MOR service	Identifies 'verb' and validates URL string for correct arguments; confirms validity of metadata format type 'metadataPrefix'.
4.	MOR service	Retrieves all metadata document identifiers from the Metacat server.
5.	MOR service	For each document identifier, request metadata record document type from Metacat server.
6.	MOR service	For each document identifier and document type that meets the metadata format ('metadataPrefix') criteria, request metadata document from the Metacat server; performs metadata translation to export metadata format; compose OAI-PMH record; add record to OAI-PMH record 'list'.
7.	MOR service	Returns OAI-PMH record 'list' to requesting harvester as an XML-encoded byte stream.

Scenario Extensions: None

Step	Condition	Action Description

Scenario Variations

Step	Actor	Action Description

4.1.6 ListSets

Characteristics Information:

Goal in Context:	Returns a list of set structures supported by the MOR service to the requesting harvester.
Scope:	
Level:	
Pre-Condition:	MOR service is listening for HTTP URL requests.
Success End Condition:	The correct list of set structures is returned to the requesting harvester.
Failed End Condition:	A list of set structures is not returned or the incorrect list of set structures is returned to the requesting harvester.
Primary Actor:	MOR service
Trigger Event:	The requesting harvester issues an HTTP URL 'ListSets' verb request.

Main Success Scenario:

Step	Actor	Action Description
1.	Harvester	The requesting harvester issues an HTTP URL 'ListSets' verb request.
2.	MOR service	Accepts and parses request.
3.	MOR service	Identifies 'verb' and validates URL string for correct arguments.
4.	MOR service	Composes 'list' of supported set structures.
5.	MOR service	Returns 'list' to requesting harvester as an XML-encode byte stream.

Scenario Extensions: None

Step	Condition	Action Description

Scenario Variations

Step	Actor	Action Description

4.2 Harvester

The following use case scenario describes the high-level interactions between a remote OAI-PMH repository and a local Metacat OAI-PMH Harvester (MOH). In this use case, the 'repository' is represented by a remote service, while the 'harvester' is represented by the MOH client.

Characteristics Information:

Goal in Context:	Harvest records from a remote repository into the database of a local Metacat instance.
Scope:	
Level:	
Pre-Condition:	Remote repository service is listening for HTTP URL requests.
Pre-Condition:	The base URL of the remove repository service is known.
Pre-Condition:	The 'metadataPrefix' value of the records to be harvested from the

	remote repository is known.
Success End Condition:	The correct list of identifiers (as determined by the 'ListIdentifiers' verb) and the correct content of metadata records (as determined by the 'GetRecord' verb) are returned by the remote repository in response to the HTTP URL requests of the MOH.
Failed End Condition:	The remote repository fails to respond to the 'ListIdentifiers' or 'GetRecord' HTTP URL requests of the MOH, or it returns an incorrect response.
Primary Actor:	MOH client
Trigger Event:	The MOH is executed from the command line by a Metacat administrator, or is launched automatically by a job scheduling application such as 'cron'.

Main Success Scenario:

Step	Actor	Action Description
1	MOH client	The MOH client is executed.
2	MOH client	Logs into Metacat using LDAP username and password (as passed to the program via command-line options.)
	MOH client	Issues 'ListIdentifier' verb request to the remote repository service.
2	Remote repository service	Accepts and parses HTTP URL 'ListIdentifier' request and responds appropriately.
	MOH client	Compares records in 'ListIdentifier' response to records in Metacat, comparing datestamps to determine whether insert, update, or delete operation should be performed. Issues 'GetRecord' verb request to the remote repository service for records that are to be inserted or updated in Metacat.
3.	Remote repository service	Accepts and parses HTTP URL 'GetRecord' requests and responds appropriately by returning metadata content.
4.	MOH client	Inserts/updates/deletes records into/from Metacat as appropriate.
5	MOH client	Logs out from Metacat and terminates execution.

Scenario Extensions: None

Step	Condition	Action Description

Scenario Variations

Step	Actor	Action Description
1	MOH Client	Variations in MOH client execution are controlled by the use of command-line options such as: <-from date> <-until date> <-metadataPrefix prefix> <-setSpec setName>

5 Development Plan

5.1 Metacat SVN Integration

All project related artifacts, including source code, templates, examples, and/or

documentation will be integrated directly with the ecoinformatics.org Metacat project from the project beginning. As such, all artifacts will be checked into and managed by the ecoinformatics.org Subversion version control system server (https://code.ecoinformatics.org) and under the “code/metacat/trunk” project directory.

A directory hierarchy follows:

- for design/planning documents - metacat/docs/dev/oaipmh
- for shell scripts and other resources - metacat/lib/oaipmh
- for Java source code - metacat/src/edu/ucsb/nceas/metacat/oaipmh

5.2 Design Details

5.2.1 Crosswalks

5.2.1.1 EML to DC crosswalk

The following table summarizes the element mappings of the EML to DC crosswalk, including notes specific to each element mapping.

EML Element	DC Element	Notes
title	title	One-to-one mapping of content
creator	creator	Use only the creator's name (givenName and surName elements); could be an organization name
keyword	subject	One subject element per keyword element
abstract	description	Must extract text formatting tags
publisher	publisher	Use only the publisher's name (givenName and surName elements); could be an organization name
associatedParty	contributor	Use only the party's name (givenName and surName); could be an organization name
pubDate	date	One-to-one mapping
dataset | citation | protocol | software	type	Type value is determined by the type of EML document rather than by a specific field value
physical	format	Use a mime type as the Format value? For example, if EML has <textFormat> element within <physical>, then use 'text/plain' as the Format value?
(1) packageId; (2) URL to the EML document	identifier	packageId can be used as the value of one identifier element; a second identifier element can hold a URL to the EML document
dataSource	source	Use the document URL of the referenced data source?
citation	relation	Use the document URL of the referenced citation?
geographicCoverage	coverage	Add separate coverage elements for geographic description and geographic bounding coordinates. For bounding coordinates, use minimal labeling, for example: `81.505000 W, 81.495000 W,`

		31.170000 N, 31.163000 N
taxonomicCoverage	coverage	Use only genus/species binomials; place each binomial in a separate coverage element
temporalCoverage	coverage	Include begin date and end date when available. For example: 1915-01-01 to 2004-12-31
intellectualRights	rights	Must extract text formatting tags

5.2.2 Repository

The Metacat OAI-PMH Repository service is implemented using the Online Computer Library Center (OCLC) OAICat Open Source Software as the basis for the implementation, with substantial customizations and modifications added to facilitate integration with Metacat.

Customizations and additions to the set of OAICat classes are described in the following table:

Package Name	Class Name(s)	Modified Class vs. New Class	Description
edu.ucsb.nceas.metacat.oaipmh.provider.server	OAIHandler.java	Modified	OAIHandler is the primary servlet class for OAICat. It has been customized to allow loading of Metacat configuration values.
edu.ucsb.nceas.metacat.oaipmh.provider.server.catalog	MetacatCatalog.java	New	MetacatCatalog is an implementation of AbstractCatalog interface. It is responsible for determining which documents exist in Metacat, their document types, and their datestamps.
edu.ucsb.nceas.metacat.oaipmh.provider.server.catalog	MetacatRecordFactory.java	New	MetacatRecordFactory, a subclass of RecordFactory, converts native Metacat documents to OAI-PMH records.
edu.ucsb.nceas.metacat.oaipmh.provider.server.crosswalk	Eml200.java Eml201.java Eml210.java	New	The set of Eml2xx.java classes are responsible for retrieving EML documents in their native format. Although these classes are subclasses of the Crosswalk class, no actual XSLT transformation is performed by them.
edu.ucsb.nceas.metacat.oaipmh.provider.server.crosswalk	Eml2oai_dc.java	New	Eml2oai_dc, a subclass of Crosswalk, is responsible for retrieving EML 2.x.y documents and transforming them to oai_dc (Dublin Core) format.

With the exception of OAIHandler.java, all other native OAICat classes are used in their original (non-modified) form and are contained in the Java archive file:

metacat/lib/oaipmh/oaicat.jar

The XSLT stylesheets used by the Eml2oai_dc.java class reside in files:

metacat/lib/oaipmh/eml200toDublinCore.xsl
metacat/lib/oaipmh/eml201toDublinCore.xsl
metacat/lib/oaipmh/eml210toDublinCore.xsl

Design Issues

1. 'Deleted' Status

OAI-PMH repositories can optionally flag records with a 'deleted' status, indicating that a record in the metadata format specified by the `metadataPrefix` is no longer available. Since the Metacat database does not provide a readily apparent mechanism for retrieving a list of deleted documents, the use of the 'deleted' status is not supported in this implementation of the OAI-PMH repository. This represents a possible future enhancement to the repository design.

2. Sets

OAI-PMH repositories can optionally support set hierarchies. Since it has not been determined how set hierarchies should be structured in Metacat, this implementation of the OAI-PMH repository does not support set hierarchies. This represents a possible future enhancement to the repository design.

3. Datestamp Granularity

When expressing datestamps for repository documents, OAI-PMH allows two levels of granularity – ***day granularity*** and ***seconds granularity***. Since the Metacat database stores the value of its `xml_documents.date_updated` field in day granularity, that is the level that is supported by the Metacat OAI-PMH Repository.

5.2.3 Harvester

The Metacat OAI-PMH Harvester client is implemented using OCLC's OAIHarvester2 open source code as its base implementation, with customizations and additions as needed to support integration with Metacat.

Customizations and additions to the set of OAIHarvester2 classes are described in the following table:

Package Name	Class Name(s)	Modified Class vs. New Class	Description

edu.ucsb.nceas.metacat.oaipmh.harvester	OaipmhHarvester.java	New	OaipmhHarvester is the primary Java class for executing the code, containing the main() method. It is a heavily modified version of the RawWrite.java class included in the OAIHarvester2 source code.
edu.ucsb.nceas.metacat.oaipmh.harvester	HarvesterVerb.java	Modified	HarvesterVerb is a parent claass to the six verb subclasses. This is a heavily modified version of the original HarvesterVerb class in the OAIHarvester2 source code.
edu.ucsb.nceas.metacat.oaipmh.harvester	GetRecord.java ListIdentifiers.java	Modified	These two verb classes are used to drive the harvest engine. The program first runs the 'ListIdentifiers' verb to determine which records exist in the remote respository, as well as their datestamps. Then it runs the 'Get Record' verb, as needed, to retrieve individual records from the remote respository for insertion/update into Metacat.
edu.ucsb.nceas.metacat.oaipmh.harvester	Identify.java ListRecords.java ListSets.java ListMetadataFormats.java	Unmodified	These verb classes are essentially unmodified from their corresponding classes in the OAIHarvester2 source code. They are not actively used by the Metacat OAI-PMH Harvester code, but are included with the source code for completeness and for potential future development.

Design Issues

1. Dryad Identifiers

Dryad stores documents using identifiers like the following:

oai:dryad-dev.nescent.org:10255/dryad.12

When harvested into Metacat, these identifiers are converted to a Metacat-legal equivalent, for example:

10255-dryad.12

(Note the use of a '-' character in place of a '/' character. It was discovered during development that the '/' character caused errors when used in Metacat identifiers.)

2. Handling of 'Deleted' status

The Metacat OAI-PMH Harvester program *does* check to see whether a 'deleted' status is flagged for a harvested document, and if it is, the document is correspondingly deleted from the Metacat repository.

3. Datestamp Granularity

When expressing datestamps for repository documents, OAI-PMH allows two levels of granularity – ***day granularity*** and ***seconds granularity***. Since the Metacat database stores the value of its 'xml_documents.last_updated' field in day granularity, that is the level that is supported by both the Metacat OAI-PMH Repository and the Metacat OAI-PMH Harvester. This has implications when Metacat OAI-PMH Harvester (MOH) interacts with the Dryad repository, which stores its documents with seconds granularity. For example, consider the following sequence of events:

1. On January 1, 2010, MOH harvests a document from the Dryad repository with datestamp '2010-01-01T10:00:00Z', and stores its local copy with datestamp '2010-01-01'.

2. Later that same day, the Dryad repository updates the document to a newer revision, with a new datestamp such as '2010-01-01T20:00:0Z'.

3. On the following day, MOH runs another harvest. It determines that it has a local copy of the document with datestamp '2010-01-01' and *does not* re-harvest the document, despite the fact that its local copy is not the latest revision.

6 Schedule

6.1 Cross-Walks

Development of the EML/DC cross-walks will require approximately 1 week. Development of the EML/Dryad Application Profile cross-walks will require 3 weeks.

6.2 Repository

Development of the OAI-PMH Metacat Repository service interface will require approximately 2 months.

6.3 Harvester

Development of the OAI-PMH Metacat Harvester service interface will require approximately 1 month.

7 Generalized Definitions

7.1 OAI-PMH

(*see OAI-PMH specification Section 2 for details - http://www.openarchives.org/OAI/ openarchivesprotocol.html# DefinitionsConcepts)*

- **Datestamp** – a datestamp is an optional construct for categorizing or grouping items based on a time frame for the purpose of "selective harvesting"; datestamps use a "day" granularity and can specify either or both beginning and ending values; datestamps can be used to identify creation, modified, and deletion events associated with an "item".
- **Harvester** – a client application that use the OAI-PMH methods to retrieve metadata from and/or information about a "repository".
- **Item** – an informational container of metadata within a "repository" that can serve as the origin of content when constructing an OAI-PMH metadata record; each "item" is considered unique within the "repository".
- **Record** – metadata expressed as a single format and as an XML-encoded stream that is returned in response to an OAI-PMH request; each record consists of a "header", "metadata", and "about" (optional) sections. The concept of "deleted records" must be supported by the "repository".
- **Repository** – a "network accessible server" that hosts metadata and complies with the OAI-PMH standard for serving metadata requests. Note that "provider" is the preferred informal name for the OAI Repository service.
- **Selective Harvesting** – selective harvesting allows a "harvester" to limit harvest requests to subsets of the metadata available from a "repository" based on either "datestamps" and/or "sets".
- **Set** – a set is an optional construct for categorizing or grouping items based on a logical theme for the purpose of "selective harvesting"; a set can be flat or hierarchical.
- **Unique Identifier** – a unique identifier (UID) unambiguously identifies an "item" in a repository; the UID must conform to the Uniform Resource Identifier (IETF RCC 2396) syntax (http://www.ietf.org/rfc/rfc2396.txt?number=2396).

8 Informational Links

- **Knowledge Network for Biocomplexity** (KNB) (NSF DEB99–80154) – http://knb.ecoinformatics.org.
- **Metacat** – http://knb.ecoinformatics.org/software/metacat/.
- **Ecological Metadata Language** (EML) – http://knb.ecoinformatics.org/software/eml/.
- **LTER Data Catalog** – http://metacat.lternet.edu/.
- **Open Archives Initiative** – http://www.openarchives.org/.
- **Dublin Core Metadata Initiative** – http://dublincore.org/.

9 OAI-PMH Error Codes

Error Code	*Description*	*Applicable Verbs*
badArgument	The request includes illegal arguments, is missing required arguments, includes a repeated argument, or values for arguments have an illegal syntax.	*all verbs*
badResumptionToken	The value of the resumptionToken argument is invalid or expired.	ListIdentifiers ListRecords ListSets
badVerb	Value of the verb argument is not a legal OAI-PMH verb, the verb argument is missing, or the verb argument is repeated.	N/A
cannotDisseminateFormat	The metadata format identified by the value given for the metadataPrefix argument is not supported by the item or by the repository.	GetRecord ListIdentifiers ListRecords
idDoesNotExist	The value of the identifier argument is unknown or illegal in this repository.	GetRecord ListMetadataFormats
noRecordsMatch	The combination of the values of the from, until, set and metadataPrefix arguments results in an empty list.	ListIdentifiers ListRecords
noMetadataFormats	There are no metadata formats available for the specified item.	ListMetadataFormats
noSetHierarchy	The repository does not support sets.	ListSets ListIdentifiers ListRecords