



GBIF Metadata Catalogue

Technical Specification

Version 0.5

Released December 2010

Suggested citation:

GBIF (2010). GBIF Metadata Catalogue Technical Specification, version 0.5 released on 22 December 2010, (contributed by Tim Robertson, Éamonn Ó Tuama, Federico Mendez), Copenhagen: Global Biodiversity Information Facility, XXpp, accessible online at http://links.gbif.org/gbif_metadata_catalogue_specification.pdf

Language of the document: English

Copyright © Global Biodiversity Information Facility, 2010

License:



This document is licensed under Creative Commons Attribution 3.0

GBIF Metadata Catalogue -Technical Specification

This document describes the functionality of the GBIF Metadata Catalogue and provides details of its implementation.

The intended audiences for this document include technical personnel involved in metadata sharing networks and developers joining the project.

The reasoning behind certain functionality is captured so that others may benefit from previous discussion outcomes and better understand the rationale for the design.

Within this document, terms in CAPITAL CASE are considered significant actors, operations or states, and the vocabulary used is of importance. The terms ORGANISATION and INSTITUTION are used interchangeably and can be considered synonymous within the scope of this document; this means a GBIF PARTICIPANT NODE, a museum or a herbarium can be considered as an INSTITUTION or an ORGANISATION within the scope of this document.

Version history

Version	Description	Date	Authors
0.1	Initial draft, detailing the functionality under development for the first implementation.	06/10/2010	Tim Robertson
0.2	Technical architecture overview	25/10/2010	Federico Mendez
0.3	Review of draft; completed sections 2.1, 2.2, 2.3, 2.4, 2.5, 3.1.2	28/10/2010	Éamonn Ó Tuama
0.4	General revision/refactoring.	07/12/2010	Éamonn Ó Tuama
0.5	Implementation view improvement	17/12/2010	Federico Mendez

Table of Contents

1. Introduction	6
1.1. Overview	6
1.2. Metadata sharing network topology	7
1.3. Alternative Implementation Options.....	7
1.4. Formats.....	8
1.5. GBIF Metadata Profile	8
1.6. Open source license	9
2. Architecture	9
2.1. Logical View	10
2.2. Use-Case View	11
2.2.1. Use Case: Search datasets by human agent	11
2.2.1.1. Full Text Search	12
2.2.1.2. Advanced Search	12
2.2.2. Use case - search datasets by computer agent	12
2.2.3. Use case - harvest metadata	12
2.3. Implementation View	14
2.3.1. Registry of Sources	15
2.3.2. Transformations	15
2.3.3. Record deletion	16
2.3.4. OAI-PMH Harvester	16
2.3.5. OAI-PMH Service	17
2.3.6. Solr Search Index.....	18
2.3.6.1. Solr Schema	19
2.4. User Interface Component	20
2.5. Deployment View	21
2.5.1. Installation and Configuration	24
2.5.1.1. Installing Apache Maven	24
2.5.1.2. Installing Solr with Apache Tomcat	24
2.5.2. Installing the metacatalogue harvester	24
2.5.2.1. Configuring Solr to index the harvested files.....	25
2.5.2.2. Running the harvester and Solr import handlers.....	25
3. References.....	27
4. Glossary of Terms.....	27

List of Tables

Table 1. Metadata format inputs to the GBIF catalogue and their transformation to DC and ISO 19139.....	15
Table 2. The three nodes of the GBIF metadata harvesting and catalogue system, and their respective requirements.....	22
Table 3. The main software components of the GBIF metadata harvesting and catalogue system.	23

List of Figures

Figure 1: The GBIF metadata network	7
Figure 2 The main components of the 4+1 Model (modified from [KRU41]).....	9
Figure 3. This logical view of the system architecture presents three main functional components: a set of online accessible metadata catalogues, a metadata harvester, and the GBIF central metadata catalogue.	11
Figure 4. The search datasets use case can involve either a human agent or a computer system	11
Figure 5. The harvesting metadata use case illustrates the requirement for external computer systems to be able to locate and harvest metadata from online repositories. ..	13
Figure 6 Simplified request for GetRecord/ListRecords verbs.....	14
Figure 7: Implementation view of the metadata catalogue showing the main software components.	15
Figure 8 Harvester main classes	17
Figure 8. Deployment of the GBIF metadata catalogue system using three computer nodes: harvester server, servlet container index and servlet container application.	21
Figure 9. The file system repository structure that is created by the GBIF harvesting component.	23

1. Introduction

1.1. Overview

This GBIF Metadata Catalogue is a software component that will enable GBIF to promote and participate in the sharing of dataset-level metadata published in commonly used standards such as Ecological Metadata Language (EML) or ISO 19139. Large distributed networks such as GBIF's that bring together many providers and consumers of data can be characterised as a Service Oriented Architecture (SOA) where consumers discover providers and their services, and loosely coupled system components can be orchestrated as needed to serve particular cases. In an SOA, the key activities of inventory, discovery and access to data and services must be well coordinated through the provision of registries and metadata catalogues, and through the generation of specific indexes. Metadata are thus a central component in an expanded GBIF network.

Metadata are literally 'data about data'. They provide information on such aspects as the 'who, what, where, when and how' pertaining to a resource. In the GBIF context, resources are datasets, loosely defined as collections of related data, the granularity of which is determined by the data custodian. Metadata can be considered from the perspective of both the data producer and the data consumer. For the producer, metadata are used to document data in order to inform prospective users of their characteristics, while for the consumer, metadata are used to both discover data and assess their appropriateness for particular needs - their so-called 'fitness for purpose'.

Metadata are usually made available in two broad categories of completeness: discovery and full. Discovery level metadata typically provide a minimum of essential information to enable a user to find out if a particular dataset exists, its location and ownership, and how to obtain further information. Full metadata include additional information on such aspects as data quality and lineage (provenance) and technical details for access and exploitation.

An important goal for GBIF is to develop the infrastructure needed across its network to support the management and delivery of the highest quality metadata that will enable potential end users to easily discover which datasets are available, and, critically, to evaluate the appropriateness of such datasets for particular purposes.

1.2. Metadata sharing network topology

The metadata catalogue will primarily be used as the central catalogue in the GBIF Data Portal for the global GBIF network, which, in turn, will broker information to wider initiatives such as EuroGEOSS¹. The metadata catalogue will support open data exchange protocols, in particular, the Open Archives Initiative for Metadata Harvesting (OAI-PMH), and therefore offer the possibility of integration with other metadata network catalogues such as Metacat², Mercury³ and Geonetwork⁴.

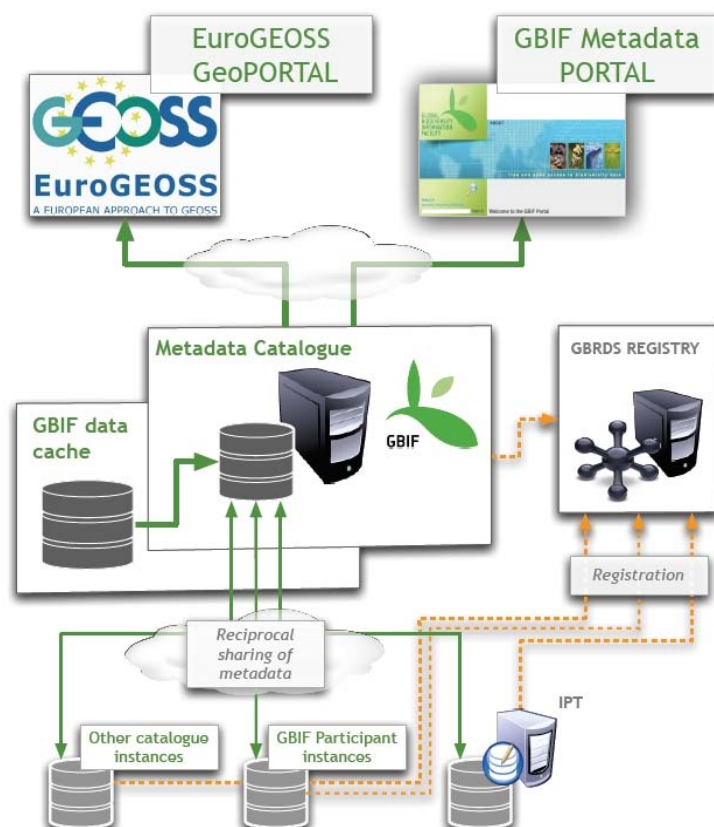


Figure 1: The GBIF metadata network

1.3. Alternative Implementation Options

In preparation for developing the metadata catalogue, GBIF evaluated the suitability of three major metadata catalogue systems (Metacat, GeoNetwork, Mercury). The findings are available as a separate report [contact authors]. In brief, at the time of evaluation (Q4 2009), Mercury was a relatively new system and not well documented, while both Metacat

¹ www.eurogeoss.eu

² <http://knb.ecoinformatics.org/index.jsp>

³ <http://mercury.ornl.gov/clearinghouse/>

⁴ <http://www.fao.org/geonetwork/srv/en/main.home>

and GeoNetwork offered well established systems with associated support communities. In the end, as the sophisticated functionality of these systems brought added complexity, especially when trying to integrate them with the GBIF portal code, GBIF opted, instead, to go for a simple solution based on assembling existing components such as the OAICat⁵ for harvesting / serving metadata and Apache Solr⁶ as the indexing/search engine.

1.4. Formats

Several metadata standards exist, either with a regional/country focus or targeted to particular communities of practice. Examples of the former include those from the Australia New Zealand Spatial Information Council (ANZLIC), Comité Européen de Normalisation (CEN) and the Federal Geographic Data Committee (FGDC), while the latter include the Dublin Core Metadata Initiative (DCMI) and Ecological Metadata Language (EML). Now, however, there is a strong promotion by most countries of the ISO 19115/19139 standards for geographic metadata, e.g., the North American Profile of ISO19115 in the USA and Canada [NAP], and the INSPIRE⁷ directive in the European Union.

1.5. GBIF Metadata Profile

A metadata profile is a recommended subset of the elements of a metadata standard for use by a particular community of users. The GBIF Metadata Profile is aimed at standardising how resources get described at the dataset level in the GBIF Data Portal. In developing this profile, the recommendations of the GBIF Metadata Implementation Framework Task Group were taken into account [MIFTG] as were requirements gathered from the community [GMP]. GBIF decided to base their profile on EML as it offered strong expressivity for describing biodiversity resources and its design was already informed by other major standards. However, GBIF does not mandate what standard a Participant should adopt; this will often be determined at a national or thematic level. Rather, in line with the MIFTG recommendations, the GBIF catalogue will accept and store metadata in all major standards, map to a common search model while avoiding lossy conversions and always return the original metadata document. To support interoperability and searching across broader metadata networks, the GBIF profile can be transformed to other formats such as ISO19139 in the case of the EuroGEOSS broker. In some cases, depending on the original format, this may result in a reduced amount of information but it should be sufficient for high-level searches and the original metadata documents will always be

⁵ <http://www.oclc.org/research/activities/oaicat/default.htm>

⁶ <http://lucene.apache.org/solr/>

⁷ <http://www.inspire-geoportal.eu/>

available. The user is referred to the GBIF Metadata Profile Reference Guide for a description of constituent elements [GMPG]

1.6. Open source license

The GBIF metadata catalogue integrates existing open source products such as Apache Solr and is itself released under an open license. It may be used and customized but support at this point is limited to installation instructions and basic operational guidelines.

2. Architecture

This section provides an overview of the architecture of the metadata catalogue RC1. Using the “4+1” View Model of architecture (Figure 2), it presents various aspects of the system through different architectural views which are intended to capture and convey to different stakeholder groups the significant architectural decisions which have been made about its design. These decisions can be organised around four views (Logical, Implementation, Process, Deployment) and the resulting architecture can be illustrated through a fifth unifying view involving selected use-cases/scenarios, hence the “4+1” moniker [KRU41].

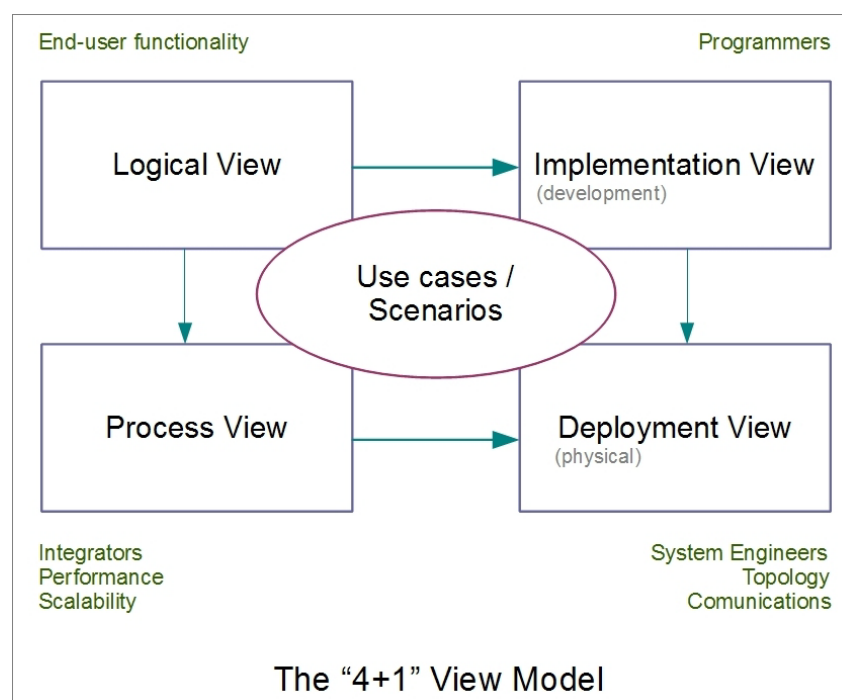


Figure 2 The main components of the 4+1 Model (modified from [KRU41])

Each of the views explains the system architecture from a particular perspective, and for a particular audience:

- *Logical View* presents the functionality that the system provides from an end-user perspective.
- The *Implementation View* (or development) provides a view of the software organisation from a programmer's perspective.
- The *Deployment View* (or physical) presents the mapping of the software components to the hardware from a systems engineer's perspective.
- The *Process View* depicts communications between system processes, run-time behaviour, etc.
- The *Use-Cases / Scenarios View* can be used to test and validate the architectural design.

Examples of logical, implementation, deployment and use-case views are presented in this document.

2.1. Logical View

The logical view (Figure 3) describes the three main components of the architecture.

These are i) one or more online accessible catalogues provided by metadata publishers and exposed using a standard interface, ii) metadata harvester(s) that look-up catalogue access points via a registry and retrieve their metadata, iii) a central metadata catalogue consisting of an indexing/search engine that indexes the metadata documents retrieved by the harvester and provides an API that allows querying of the indexed documents.

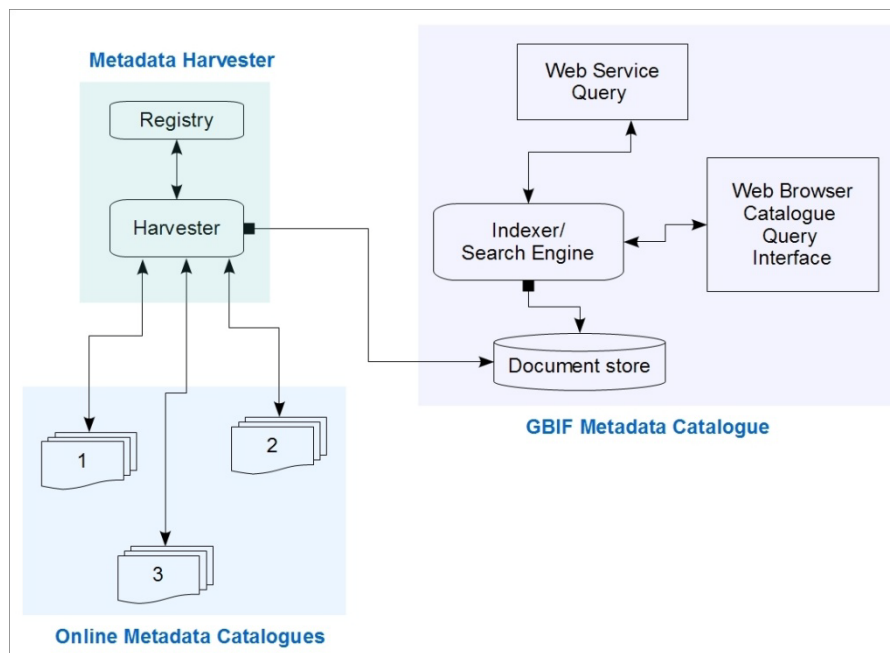


Figure 3. This logical view of the system architecture presents three main functional components: a set of online accessible metadata catalogues, a metadata harvester, and the GBIF central metadata catalogue.

2.2. Use-Case View

The Use-Case view presents use cases or scenarios that describe some aspect of the system in use. Use-case diagrams are used to provide a graphical view of some of the actors and actions involved. We describe here two use cases: i) searching for datasets and ii) harvesting metadata.

2.2.1. Use Case: Search datasets by human agent

The *search datasets* use case (Figure 4) depicts a requirement for a human agent to be able to search for datasets using the metadata catalogue system. To enable this, the system provides two main search functionalities: Full Text Search and Advanced Search. For both functionalities the system will display a list of data sets containing the following information: title, provider, description (abstract) and hyperlink to view the full metadata document in the original format (DIF, EML, etc.) provided by the source.



Figure 4. The search datasets use case can involve either a human agent or a computer system

2.2.1.1. Full Text Search

The user enters a string, the string consist of one or more tokens. A token can be a single word (or term) or a phrase. A term may contain wildcards in the middle or at the end (* for matching zero to many characters, ? for matching zero or one character). The system searches for the string in all document fields.

2.2.1.2. Advanced Search

In advanced search, the user searches for datasets by providing several search criteria that are input via the advanced search form. The form consists of three main sections:

- *Text Search*: allows the user to select a predefined field (text, title and provider) and indicate a search string for the selected field; additionally the user can indicate the Boolean operator between the search fields (AND and OR). The final search query will be interpreted by the order of fields; the last Boolean operator is omitted (since it doesn't affect the query).
- *Date Search*: provides two input calendar fields: start date and thru date. The system will search for the documents whose start and end dates are between the parameters start and thru date. In some cases metadata formats only provide one date (Dublin Core for example), so the system will use that date for both start and end date.
- *Geographic Search*: allows the user to input geographic bounding coordinates in order to restrict the search results. The system allows the user to enter the values for East, West, North and South bounding coordinates. Additionally, the user can use a map (OpenLayers, <http://openlayers.org/>) to select a square region to indicate the bounding coordinates.

2.2.2. Use case - search datasets by computer agent

There is also a requirement for external computer systems to be able to search the metadata catalogue (Figure 4). A computer agent submits a REST-based query consisting of a URI with appended set of key-value pairs [see separate document on the Solr syntax for constructing REST-based queries]. In response, the system returns an XML formatted list of dataset items each containing the following information: title, provider, description (abstract) and hyperlink to view the full metadata document in the original format (DIF, EML, etc.) provided by the source.

2.2.3. Use case - harvest metadata

The *harvest metadata* use case (Figure 5) illustrates a requirement for an external computer system to be able to locate and harvest (retrieve) metadata from online metadata repositories. This is related to the search use case but is not as refined, offering far fewer criteria for filtering what is retrieved (e.g., all documents available in a particular metadata format).

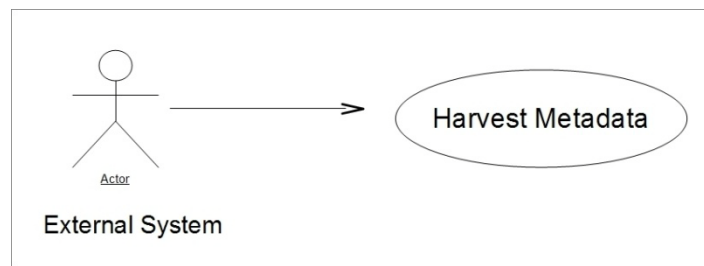


Figure 5. The harvesting metadata use case illustrates the requirement for external computer systems to be able to locate and harvest metadata from online repositories.

The sequence of events and actors associated with the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) GetRecord / ListRecords requests are depicted in Figure 6.

Open Archives Initiative Protocol for Metadata Harvesting

The OAI-PMH provides a low-barrier mechanism for interoperability across distributed metadata repositories. Data providers expose metadata through a service; consumers aggregate metadata through a client application known as a harvester that issues OAI-PMH service requests over HTTP. The standard defines six requests (verbs) that enable an agent to access an online repository and retrieve its metadata⁸.

The OAI-PMH specification stipulates that a parameter “metadataPrefix” (to indicate the format of the metadata requested) must be specified in the request. The system uses this parameter to take one of the following decisions:

- The metadataPrefix is not supported.
- The metadataPrefix and the record source format are the same, so the record as stored in the file system can be returned.
- There exists an XSLT transformation (using an XSLTCrosswalk) from the source record format to the format specified by the metadataPrefix.

⁸ 1. GetRecord (return individual record) 2. Identify (retrieve information about repository) 3. ListIdentifiers (retrieve headers of records); 4. ListMetadataFormats (return metadata formats available) 5. ListRecords (return records from repository) 6. ListSets (retrieve set structure (groupings) of repository).

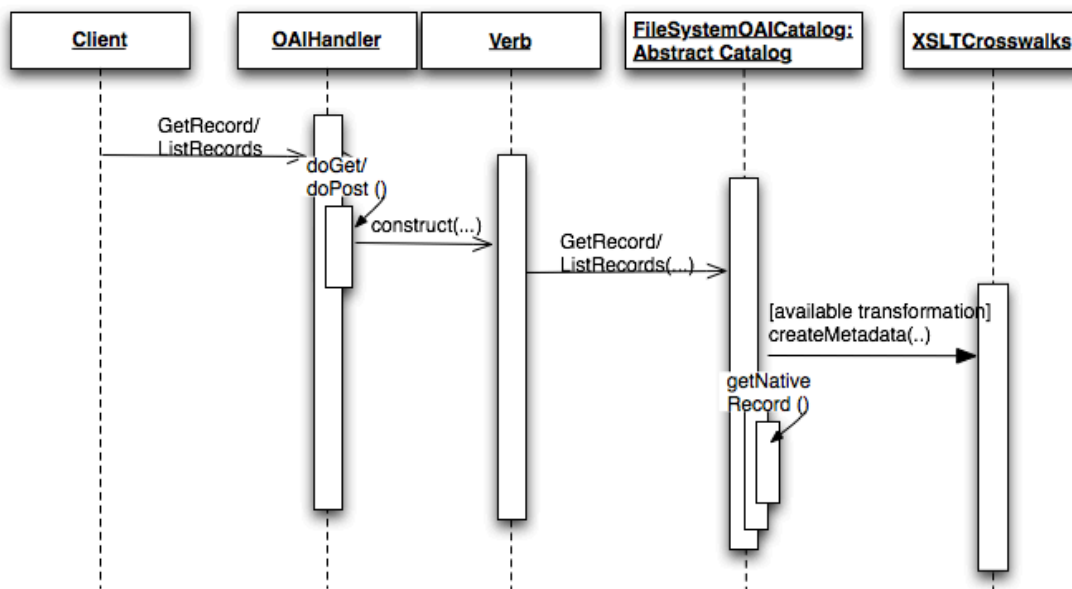


Figure 6 Simplified request for GetRecord / ListRecords verbs.

The GBIF metadata catalogue service undertakes both harvesting and serving roles; aggregating metadata from other OAI-PMH repositories and serving metadata via OAI-PMH to other harvesting services such as the EuroGEOSS broker. The harvested metadata are stored in a local file system. The system can apply XSLT transformation to create a new document based of the content of the existing one (e.g., transforming an EML document to an ISO19139 one).

2.3. Implementation View

The Implementation View depicts the software used to implement the system. The GBIF metadata catalogue is built upon the existing open source tools. The main components are illustrated in Figure 7 and described briefly in the following sections.

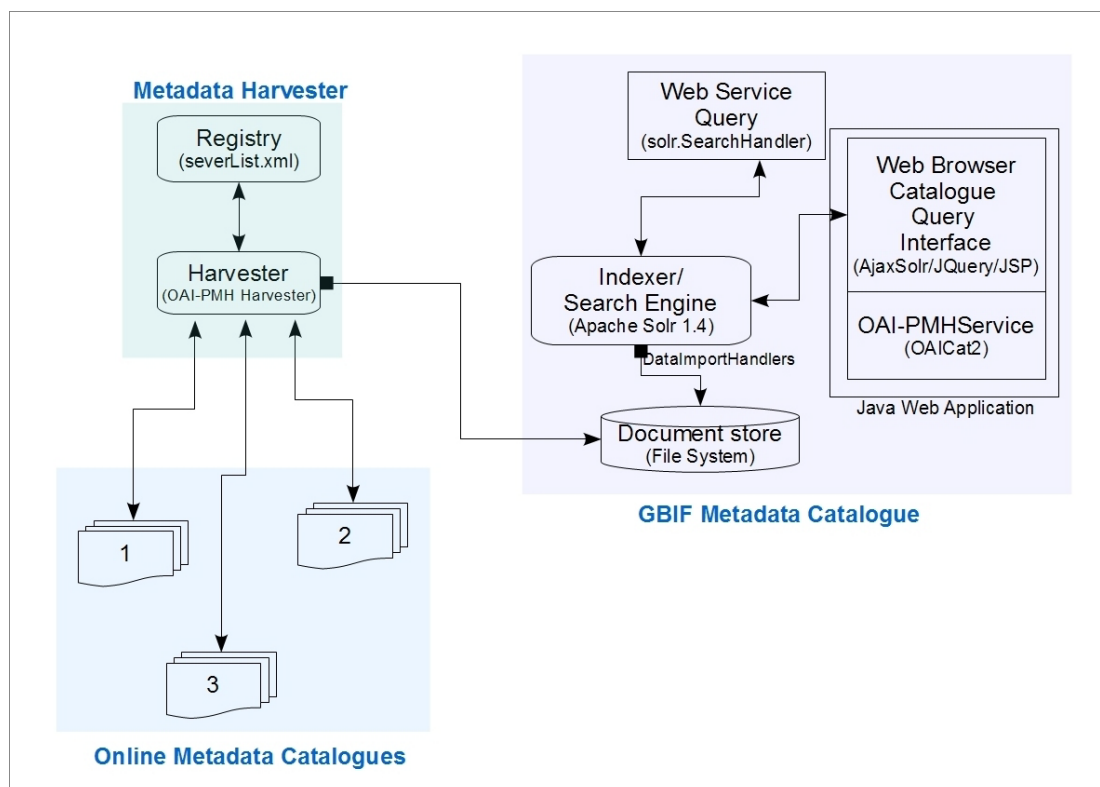


Figure 7: Implementation view of the metadata catalogue showing the main software components.

2.3.1. Registry of Sources

At the time of writing the Registry of sources is maintained in a simple XML file. This allows for ease of adoption of the catalogue by others managing small networks. It will, in time, be replaced with an adapter to the GBIF Registry to provide the location and types of endpoints to harvest. The XML file format is explained in section 2.5.2.2.

2.3.2. Transformations

The input formats to the metadata catalogue dictate the transformations required. The GBIF catalogue currently accepts the following metadata input formats: EML (v2.0.1; v21.0; v2.1.1); ISO 19139; DC, DIF. All are available for onward harvesting in their native formats but certain transformations to DC and ISO 19139 are also available via XSLT. The following table summarises the transformations currently supported from input to the DC and ISO 19139 outputs.

Table 1. Metadata format inputs to the GBIF catalogue and their transformation to DC and ISO 19139.

Input	DC Output	ISO 19139 Output
ISO 19139	yes (lossy)	no transformation required

EML v2.0.1	yes (lossy)	yes - only those terms found in the GBIF Metadata Profile
EML v2.1.0	yes (lossy)	yes - only those terms found in the GBIF Metadata Profile
DIF v9.7.1	yes (lossy)	no current support
DC	no transformation required	no current support

2.3.3. Record deletion

OAI-PMH provides a means to indicate record deletion, although implementations are not mandated to support it. Without signaling record deletion, to determine if a resource (i.e. set of metadata records) has been fully harvested one must either:

- Issue a “get by id” for each record to determine if it still exists. This strategy can have a significant performance overhead for large resources.
- Be sure to complete a full resource harvest, updating some flag (e.g. update a timestamp) on each record, and then identify those records that have not been flagged. This strategy has risks involved if a harvest is interrupted for some undetected reason as subsets of data might incorrectly be flagged for deletion in error. This may be mitigated by vigilance in catching issues during harvesting.

The GBIF metadata catalogue does not currently support the deletion flag; this feature will be implemented in future versions.

2.3.4. OAI-PMH Harvester

The harvester is a standalone Java application, it has two operation modes, configured using special XML files (harvester.properties and serverList.xml – see section 3.5.1.5):

- RunOnce: in this mode the harvester runs only once; during its run the server list is used to get the latest records.
- Thread Pool Server: under this mode the harvester creates a pool of threads to contact the server list, one thread being created for each server until a maximum of the configured thread pool size (see: section 2.5.2.2).
- Each server is contacted at a set interval (“pollingFrequency” seconds). The last date of harvesting is stored in the “lastDatePolling” field of the configuration file so that, the next time the server is contacted, the harvester will request the new records since the last run.

The harvester makes extensive use of the open source project “OAIHarvester2” which supports OAI-PMH v1.1 and v2.0. The source code of this project was not modified but extended to handle the harvested XML payload. The payload is delivered as a single file of aggregated xml documents (one per metadata resource). The GBIF extension extracts

individual xml metadata files and generates a list of these files. The following diagram shows the classes that compose the harvester application:

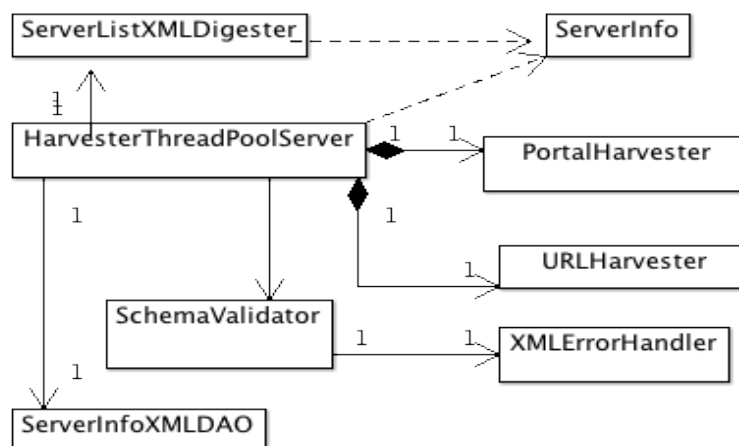


Figure 8. The main classes of the OAI-PMH Harvester.

The HarvesterThreadPoolServer class is responsible for loading the configuration files, and initialising and executing the thread pool. This class contains two inner static classes: PortalHarvester and URLHarvester. The PortalHarvester gathers the information from the GBIF portal and the URLHarvester is used to harvest the OAI-PMH services registered in the server list.

The SchemaValidator validates that each record is a well-formed XML file. If the harvested file is not a valid xml document, the XMLErrorHandler makes a copy the document in the "baseErrorDirectory" folder (see: section 2.5.2.2). After each run, the harvester stores the date when a server was contacted. Since an XML file is used to store this information and the harvester is a multiple-thread environment, the modification to the file must be synchronized. To accomplish this, the ServerInfoXMLDAO creates a singleton instance of itself and executes a "server" record (xml element) update.

The Digester framework (<http://commons.apache.org/digester/>) is used to simplify the XML parsing: the class ServerListXMLDigester processes the file serverList.xml based on parsing rules defined in "serverlist-digester-rules.xml".

2.3.5. OAI-PMH Service

This component was implemented by modifying the OAICat (<http://www.oclc.org/research/activities/oaicat/default.htm>) web application. The main changes, made to achieve specific objectives, are:

- *Dynamic load of file store.* The default behaviour of the server is to load the file list at the server start-up. Since the harvester can modify the file store, the server loads the file list every time a ListIdentifiers or ListRecords verb is requested.

- *Support multiple XSL transformations for an input format.* The reference implementation only support one transformation. A new configuration file “transformations.properties” defines the supported transformations. When a getRecord or ListRecords verb is submitted, the application will apply the appropriate process for each file in the store:
 - If the metadataPrefix requested is the same as the file format, no transformation is applied and the file is send as is.
 - If a transformation exists from the file format to the requested metadataPrefix, the transformation is applied. The available transformations are registered in the “transformations.properties” file, the following transformations are currently supported:

```
eml.iso19139=true
oai_dc.iso19139=true
dif.iso19139=true
```

The XSL transformations are configured in the oaicat.properties as follows:

```
Crosswalks.dif=org.oclc.oai.server.crosswalk.XSLTCrosswalk
XSLTCrosswalk.dif.schemaLocation=http://gcmd.gsfc.nasa.gov/Aboutus/xml/dif/
http://gcmd.gsfc.nasa.gov/Aboutus/xml/dif/dif_v9.7.1.xsd
XSLTCrosswalk.dif.xsltName=difToiso19139.xsl
```

2.3.6. Solr Search Index

Apache Solr (<http://lucene.apache.org/solr/>) is an open source enterprise search serve providing indexing services based on the Apache Lucene framework. It has numerous other features such as search result highlighting, faceted navigation, query spell correction, auto-suggest queries and “more like this” for finding similar documents. The metadata application uses Solr to provide search functionality. The XML documents gathered by the harvester are imported into Solr using specific data import handlers (<http://wiki.apache.org/solr/DataImportHandler>) for each input format. The data import handlers are implemented using three main features available in Solr:

- FileDataSource: allows fetching content from files on disk.
- FileListEntityProcessor: an entity processor used to enumerate the list of files.
- XPathEntityProcessor: used to index the XML files, it allows defining of Xpath expressions to retrieve specific elements.
- PlainTextEntityProcessor: reads all content from the data source into a single field; this processor is used to import the whole XML file into one field.
- DateFormatTransformer: parses date/time strings into java.util.Date instances; it is used for the date fields.
- RegexTransformer: helps in extracting or manipulating values from fields (from the source) using Regular Expressions.

- `TemplateTransformer`: used to overwrite or modify any existing Solr field or to create new Solr fields; it is used to create the id field.
- `org.gbif.solr.handler.dataimport.ListDateFormatTransformer`: this is a custom transformer to handle non-standard date formats that are common in input dates; it can handle dates with formats like: 12-2010, 09-1988, and (1998)-(2000). It has three important attributes: i) *separator* that defines the character/string to be used as separator between year and month fields, ii) *lastDay* to define if the date to be used with a particular year value (e.g., 1998) should be the first or the last day of the year: if the year is being interpreted as a *beginDate*, then the value is set to yyyy-01-01 and *lastDay* is set to false; if the year is interpreted as an *endDate* then the value is set to yyyy-12-31 and the *lastDay* value is set to true, iii) *selectedDatePosition* to define which date is being processed when a range of dates is present in the input field; for example:

```
<field column="beginDate"    listDateTimeFormat="yyyy-MM-dd"
selectedDatePosition="1" separator="_" lastDay="false"
xpath="/dc/date"/>
```

imports the "*dc/date*" into the begin date using "_" as separator ;
selectedDatePosition="1" states the date to be processed is the first one in the range of dates and *lastDay* is thus set to false.

The data import handlers that will be invoked by the harvester are configured in the "harvester.properties" file by setting the "SolrImportHandlers" parameter, for example, the line below configures three data import handlers (separated by ","):

```
solrImportHandlers=emlimport,dcimport,difimport
```

2.3.6.1. Solr Schema

The `schema.xml` file contains all of the details about which fields the documents can contain. The metadata application stores a subset of the available information in the metadata documents as Solr fields. Additionally a special field is used to store the whole XML document to enable full text search. The following table summarizes the structure of Solr schema file:

Field	Data type	Description
id	String	Document unique key, the full file name is used for this field.
title	Text	Title of the dataset,
provider	Text	Provider of the dataset
providerExact	String	Same as the previous field, but uses String

		data type for facets and exact match search
description	Text	Description or abstract of the dataset
beginDate	Date	Begin date of the temporal coverage of dataset
endDate	Date	End date of the temporal coverage of dataset, when the input format only supports one dataset date, beginDate and endDate will contain the same value
westBoundingCoordinate	Double	Geographic west coordinate
eastBoundingCoordinate	Double	Geographic east coordinate
northBoundingCoordinate	Double	Geographic north coordinate
southBoundingCoordinate	Double	Geographic south coordinate
fullText	Text	The complete text of the XML metadata document
externalUrl	String	Url containing specific information about the dataset; in the case of the GBIF-Data Portal, this field points to the dataset information page
serverId	String	Id of the source OAI-PMH Service; this information is taken from the file system structure and is used for the facets search.

Note: the data import handlers should be consulted for a fuller description how these fields are populated.

2.4. User Interface Component

The user interface component is a thin layer between the end user and the Solr server. It is implemented using the following technologies:

- AJAX Solr (<https://github.com/evolvingweb/ajax-solr/wiki>): this is a JavaScript framework which provides several widgets that facilitate the communication with the Solr server and minimize the required effort to build user interface components. Specifically, the components used from this library are: facets, text search and results pagination.
- JQuery/JQuery UI (<http://jquery.com/>): AJAX Solr requires a library to implement the AJAX requests. JQuery was chosen for this purpose. Additionally, several JQueryUI widgets are extensively used for a richer user experience.
- SyntaxHighlighter(<http://alexgorbatchev.com/SyntaxHighlighter/>): this is a code syntax highlighter developed in JavaScript, This component is used for displaying the XML view of a metadata document .

- HTML/JSP: While the user interface component is almost completely (ca. 95%) independent of the server-side technology, in the current implementation, two server-side services are needed:
 - Solr Server URL: this parameter is set in the web.xml file as context parameter.
 - XMLDataSetRespond: this servlet returns a formatted view of the XML metadata document displaying the file as a tree-view.

Java servlet container: the servlet component used is Apache Tomcat 6 but migrating the application to another platform is a simple task since the main functionality is implemented using JavaScript frameworks.

2.5. Deployment View

The Deployment View depicts a static view of the run-time configuration of processing nodes and the software components that run on those nodes. The deployment scenario described in this section (Figure 8) assumes the existence of three server nodes but in a minimalistic scenario the system can be deployed on just one node server.

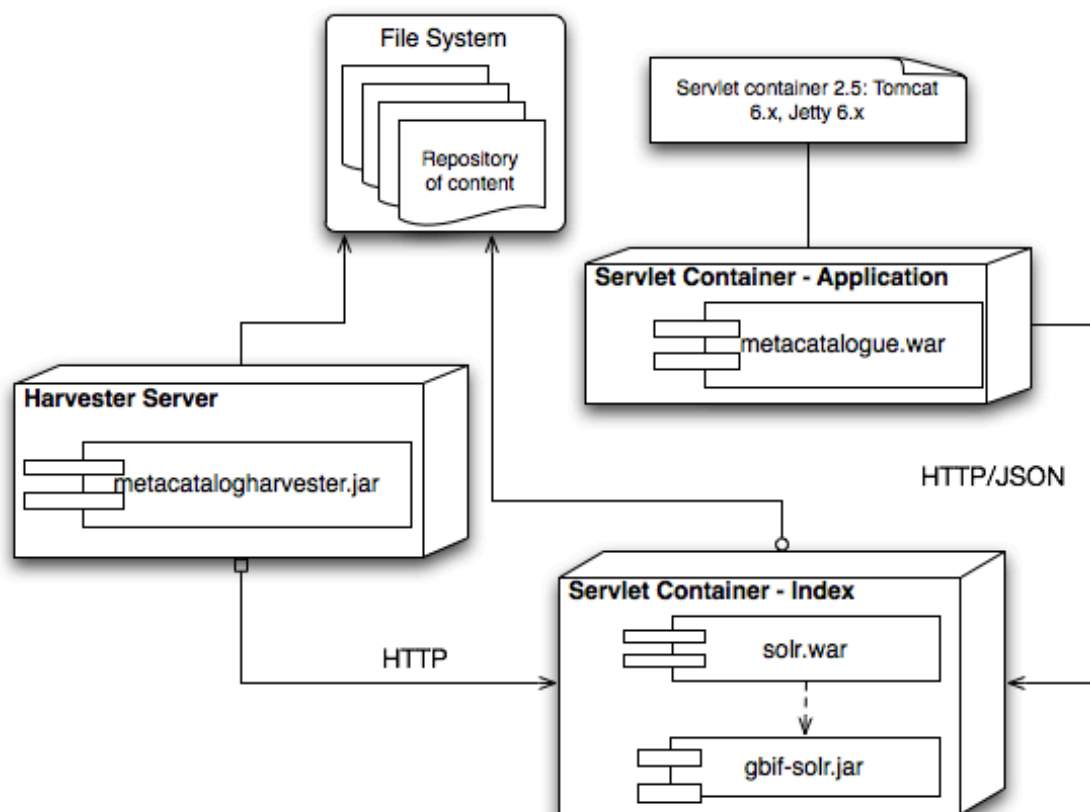


Figure 9. Deployment of the GBIF metadata catalogue system using three computer nodes: harvester server, servlet container index and servlet container application.

Descriptions of the three nodes and their software dependencies are presented in Table 2 and the individual components hosted on these nodes are described in Table 3.

Table 2. The three nodes of the GBIF metadata harvesting and catalogue system, and their respective requirements.

Name & Description	Requirements
Harvester Server - hosts the harvester application	JVM 1.6x; HTTP access to the URLs of registered metadata repositories; read/write access to the file system repository. The harvester will store the xml files in the repository.
Servlet Container - Application - hosts the main catalogue application components including search functionality, request and response handling, web interface, and the OAI-PMH Service (both were described in the Use Case View).	JVM 1.6x; Servlet 2.5 compliant container; HTTP access to the Solr Index server.
Servlet Container - Index - hosts the Solr 1.4 indexing services.	JVM 1.6x; Servlet 2.5 compliant container; read access to the file system repository.

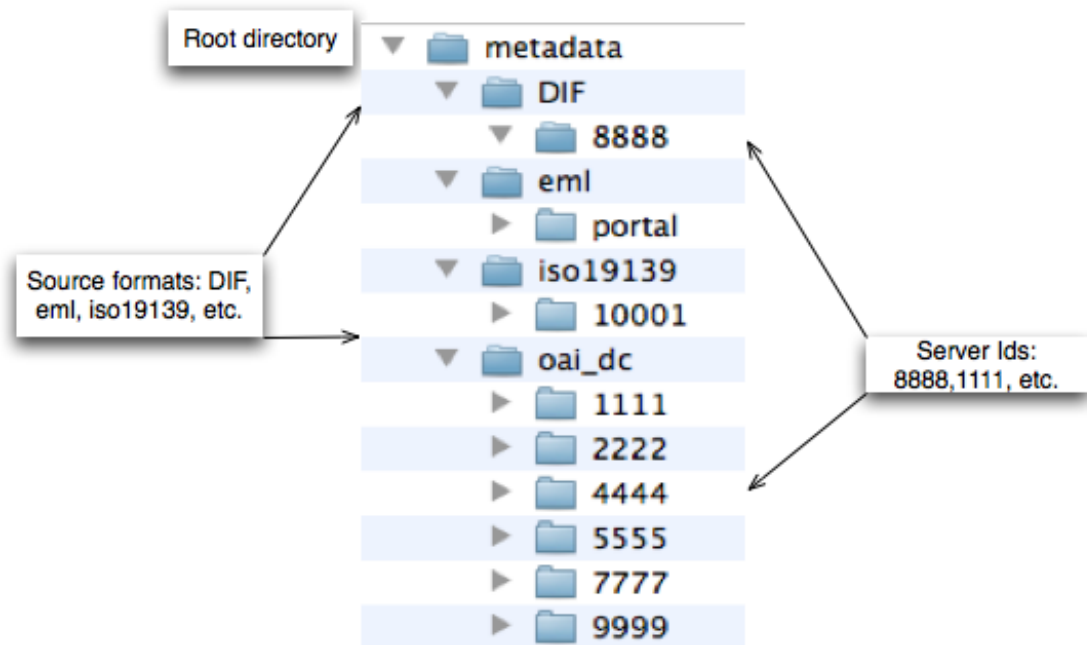


Figure 10. The file system repository structure that is created by the GBIF harvesting component.

Table 3. The main software components of the GBIF metadata harvesting and catalogue system.

Component	Description
File System Repository	A reachable file server. If the directory is empty, the first time the harvester runs, the structure depicted in Figure 9 is created inside the directory.
metacatalogue.war	A Java Web Application that includes i) the OAI-PMH Service as an implementation of the OAI-PMH repository protocol and ii) the search functionalities (both explained in more detail in the Use Case View section). This web application does not have dependencies on libraries except for those that are already included in the war file.
metacatalogharvester.jar	A standalone Java application responsible for harvesting metadata from the list of OAI-PMH servers and the GBIF portal. This application can be executed in one of two modes: "run once" and "scheduled thread pool server". The "run once" is intended for use when the operating system (or any other task scheduler service) is responsible for running the harvester

	periodically. The scheduled thread pool service is a configurable java application that allows running the harvester at fixed time intervals.
solr.war	The standard Solr 1.4 web application; it can be downloaded from the main Solr distribution repository.
gbif-solr.jar	A small Java library that contains utility classes used by the data import handlers; specifically the Dublin Core import handlers requires the class "ListDateFormatTransformer" included in this jar archive. This file must be copied to the Solr "lib" folder.

2.5.1. Installation and Configuration

This section describes the mains steps that should be followed in order to install different components of the GBIF metadata harvesting and catalogue system.

2.5.1.1. *Installing Apache Maven*

- Download the latest release of Apache Maven from <http://maven.apache.org/>

2.5.1.2. *Installing Solr with Apache Tomcat*

Download Apache Solr 1.4 from <http://lucene.apache.org/solr/>

Download Apache Tomcat 6 from <http://tomcat.apache.org/>

To install Solr with Apache Tomcat please follow the guide

<http://wiki.apache.org/solr/SolrTomcat>

2.5.2. Installing the metacatalogue harvester

- Download the source code from the svn repository: <https://gbif-metadata.googlecode.com/svn/trunk/metacatalog>; this creates a local metacatalog directory
- Enter the metacatalog directory and build and package the application using the Maven command:
 - `mvn -Dmaven.test.skip=true package`
- This command will create in the Maven *target* directory the file `metacatalog-0.0.1-SNAPSHOT.jar` and the directory *libs*; both must be copied to the harvester installation directory.

2.5.2.1. *Configuring Solr to index the harvested files*

- Stop the Solr servlet container.
- From the metacatalog project directory, copy the files dc-import, eml-import, schema.xml and solrconfig.xml into the directory SOLR_HOME/conf. (Those files are located in the solr.config directory/package)
- Modify the "baseDir" in dc-import and eml-import files, this attribute must point to the directory where oai_dc and eml files are stored in the file system.

```
<entity name="dcdataset" rootEntity="false" dataSource="null"
        processor="FileListEntityProcessor" fileName="^\.*\.xml$"
        recursive="true"
        baseDir="/opt/metacatalog/data/oai_dc">
```

- Copy the file gbif-solr.jar from the target directory to the SOLR_HOME/lib directory (this file was generated when the project was compiled using the Maven package phase in previous section).

2.5.2.2. *Running the harvester and Solr import handlers*

The metadata harvesting process can be set up to run once (and repeated manually as required) or as a scheduled task (using the scheduled thread pool server).

- For running the "run once" harvester use the following command:
 - Java -jar metacatalog-0.0.1-SNAPSHOT.jar runOnce
- For running the "scheduled thread pool server" use the following command: Java -jar metacatalog-0.0.1-SNAPSHOT.jar.
 - The "scheduled thread pool server" uses two configuration files:
 - harvester.properties: located in the directory/package org.gbif.metacatalog.harvester. This file contains the following parameters:

```
#number of threads in the pool
threadpoolsize=4
#frequency in seconds that the GBIF portal will be harvested
portalharvesterpollingfrequency=360000
#File system repository directory
outputdirectory=/opt/metacatalog/data/
#File system directory repository for the GBIF portal
portaloutputdirectory=/opt/metacatalog/data/eml/
```

- serverList.xml: located in the directory/package org.gbif.metacatalog.harvester. This file contains the list of OAI-PMH

servers that will be harvested. The structure of each element in the server list is:

```
<?xml version="1.0" encoding="UTF-8"?>
<serverlist>
  <server>
    <organisationId>1</organisationId>
    <organisation>Secretariat..</organisation>
    <serverId>1111</serverId>
    <country>Burkina Faso</country>
    <serverUrl>http://gbif.spconedd.org/oai2.php</serverUrl>
    <metadataPrefix>oai_dc</metadataPrefix>
    <pollingFrequency>360</pollingFrequency>
    <lastDatePolling></lastDatePolling>
  </server>...
```

- Organisation id: unique identifier for the organization contact (note: this is not used in the current implementation).
- ServerId: unique identifier for the source; this identifier will be used to create a directory where all the documents obtained from this source will be stored (see Figure 6).
- Country
- serverUrl: OAI-PMH service URL.
- metadataPrefix: prefix used by the server OAI-PMH protocol; this operation is repeated for each available metadata format required
- pollingFrequency: frequency in seconds at which the server will be harvested; the thread assigned for the server will be executed every "pollingFrequency" number of seconds.
- lastDatePolling: is used by the harvester to store the last date on which the source was harvested; the next time the server is harvested this value will be used to filter the data that was changed since the last run.
- Note: it is highly recommended to use the operating system scheduled task services; for example the "run Once" harvester can be used for incremental harvesting using a crontab job description as the following:
 - 0 0 * * * -jar metacatalog-0.0.1-SNAPSHOT.jar runOnce
- The harvester will invoke the Solr import handler after each run, but if you desire to invoke the handlers manually, it is possible to use the administration web

interface or any other tool able to invoke HTTP URLs (curl for example). The administration Web interface can be executed using a regular Internet browser; for example, invoking the URL:

<http://solrserver:8080/solr/admin/dataimport.jsp?handler=/emlimport>

will display a Web page that allows the user to execute several handler commands: full-import, debug, delta import, commit and execute the handler in verbose mode (see the Solr documentation for more information).

3. References

- [KRU41]: The "4+1" view model of software architecture, Philippe Kruchten, November 1995, <http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/Pbk4p1.pdf>
- [OAIPMH]: The Open Archive Initiative Protocol for Metadata Harvesting (<http://www.openarchives.org/pmh/tools/tools.php>)
- [NAP]: North American Profile of ISO19115:2003 - Geographic information - Metadata (NAP - Metadata, version 1.0.1). <http://www.fgdc.gov/standards/projects/incits-I1-standards-projects/NAP-Metadata/napMetadataProfileV101.pdf>
- [IPTMPA]: IPT Metadata Profile Additions. http://rs.gbif.org/schema/eml-gbif-profile/dev/IPT-Metadata-Profile-Additions_v3.doc
- [MIFTG]: Report of the GBIF Metadata Implementation Framework Task Group, December 2009. <http://www2.gbif.org/GBIF-MIFTG-Report.pdf>
- [GMP]: GBIF metadata profile element requirements. http://rs.gbif.org/schema/eml-gbif-profile/dev/IPT-Metadata-Profile-Additions_v3.doc
- [GMPG]: GBIF Metadata Profile - How-To Guide [URL]

4. Glossary of Terms

Term	Meaning
OAI-PMH	Open Archives Initiative - Protocol for Metadata Harvesting. http://www.openarchives.org/OAI/openarchivesprotocol.html
EML	Ecological Metadata Language. http://knb.ecoinformatics.org/software/eml/
ISO19115	ISO metadata standard for describing geographic information and services. http://www.iso.org/iso/catalogue_detail.htm?csnumber=26020
ISO19139	Geographic Metadata XML (gmd) - an XML encoding of ISO19115

	http://www.iso.org/iso/catalogue_detail.htm?csnumber=32557
DIF	Directory Interchange Format. http://gcmd.nasa.gov/User/difguide/
DC	Dublin Core. http://dublincore.org/
DCMI	Dublin Core Metadata Initiative. http://dublincore.org/
CEN	Comité Européen de Normalisation. http://www.cen.eu/cen/
FGDC	Federal Geographic Data Committee. http://www.fgdc.gov/metadata
ANZLIC	Australia New Zealand Spatial Information Council. http://www.anzlic.org.au/