Extract, Transform and Load Architecture for Metadata Collection

Marisa R. De Giusti
Com. de Investigac. Científ. de la Pcia. de Bs. As. (CICPBA)
Proyecto de Enlace de Bibliotecas (PrEBi)
Servicio de Difusión de la Creación Intelectual (SeDiCI)
Universidad Nacional de La Plata
La Plata, Argentina
marisa.degiusti@sedici.unlp.edu.ar

Néstor F. Oviedo, Ariel J. Lira
Proyecto de Enlace de Bibliotecas (PrEBi)
Servicio de Difusión de la Creación Intelectual (SeDiCI)
Universidad Nacional de La Plata
La Plata, Argentina
{nestor,alira}@sedici.unlp.edu.ar

Abstract *— Digital repositories acting as resource aggregators typically face different challenges, roughly classified into three main categories: extraction, improvement and storage. The first category comprises issues related to dealing with different resource collection protocols –OAI-PMH, web-crawling, web-services, etc.– and their representation: XML, HTML, database tuples, unstructured documents, etc. The second category comprises information improvements based on controlled vocabularies, specific date formats, correction of malformed data, etc. Finally, the third category deals with the destination of downloaded resources: unification into a common database, sorting by certain criteria, etc.*

This paper proposes an ETL architecture for designing a software application that provides a comprehensive solution to challenges posed by a digital repository as resource aggregator. Design and implementation aspects considered during the development of this tool are described, focusing especially on architecture highlights.

Keywords*:* aggregation, data integration, data warehousing, digital repositories, harvesting.

INTRODUCTION

Resource aggregation is one of the activities performed in the context of digital repositories. Its goal is usually to increase the amount of resources exposed by the repository. There are even digital repositories that are only resource aggregators – e.g. do not expose their own material.

Aggregation starts with relevant resource collection from external data sources; currently, there are several communication and transference protocols, as well as techniques for collecting available material from data sources that were not originally intended for this purpose. Some of these protocols and techniques include:

OAI-PMH [1]: A simple and easy-to-deploy protocol for metadata exchange. It does not impose restraints upon resource representation, allowing the service repository to select metadata format.

Web-Crawling [2]: A robot scans web pages and tries to detect and extract metadata. This method is useful for capturing the large volume of information distributed throughout the Web, but the problem is that documents lack homogeneous structure.

Web-Services: Using SOAP or XML-RPC as communication protocols in general,

resource structure depends on server implementation.

As briefly seen above, methods and means for resource collection vary significantly depending on the situation, context and specific needs of each institutional repository, both at the level of communication protocol and of data. Therefore, independent data collection processes and different analysis methodologies are critical to standardizing aspects such as controlled vocabularies, standard code use, etc.

Likewise, when it comes to determining the use of collected information, there are also different situations that depend on specific repository needs. The most common scenario is the unification of collected resources into a central database. Another usual approach is to logically sort information –i.e. by topic, source country, and language– inserting the resources in different data stores. Furthermore, it may be necessary to generate additional information by applying analysis and special processes to resources –concept extraction, semantic detection and ontology population, quotation extraction, etc– and use different databases to store this new information.

In general, information from different repositories is typically diverse in structure, character encoding, transfer protocols, etc., requiring different extraction and transformation approaches. Analogously, specific capabilities are required to interact with each data store that receives the transformed resources. This requires a set of organized tools that provide a possible solution. There are a number of potential complications: initially, it is necessary to find –or develop– a series of tools, each tailored to solve a specific problem, and then install, setup, test and launch each of them. Subsequently, there is the problem of tool coupling, and the need to ensure reliable interaction, since it is highly probable that these tools act upon the same dataset. On the other hand, it is important to consider the type of synchronization mechanism used to determine task sequences: order of tasks that each tool will carry out, which tasks can be executed simultaneously and which ones sequentially, etc.

From a highly abstracted viewpoint, it is possible to identify three main issues: Extract, Transform and Load (ETL).

DEVELOPMENT OF A UNIFIED SOLUTION

ETL [3] is a software architectural pattern in the area of Data Integration, usually related to data warehousing. This process involves data extraction from different sources, subsequent transformations by rules and validations, and final loading into a Data Warehouse [4] or Data Mart [5]. This architecture is used mainly in enterprise software to unify the information used for Business Intelligence [6] processes that lead to decision-making.

Given the challenges presented by resource aggregation via institutional repositories throughout the different phases in heterogeneous information management, a comprehensive ETL solution is highly practicable.

This paper presents the development of a tool intended as a unified solution for these various issues.

The design is based on the following premises:

(a) Allow the use of different data sources and data stores, encapsulating their particular logic in connectable components.

(b) Allow tool extension with new data source and data store components developed by third parties.

(c) Allow selection and configuration of the analysis and transformation filters supplied by the tool, encapsulating the particular logic in connectable components.

(d) Allow tool extension by adding new analysis and transformation filter components developed by third parties.

(e) Present an abstract resource representation for uniform resource transformation.

(f) Provide a simple and intuitive user interface for tool management.

(g) Provide an interface for collection and storage management.

(h) Achieve fault tolerance and resume interrupted processes after external problems.

(i) Provide statistic information about process status and collected information.

The software tool was developed along these premises, trying to keep components as separated/uncoupled as possible. Fig. 1 shows an architecture diagram for the tool.

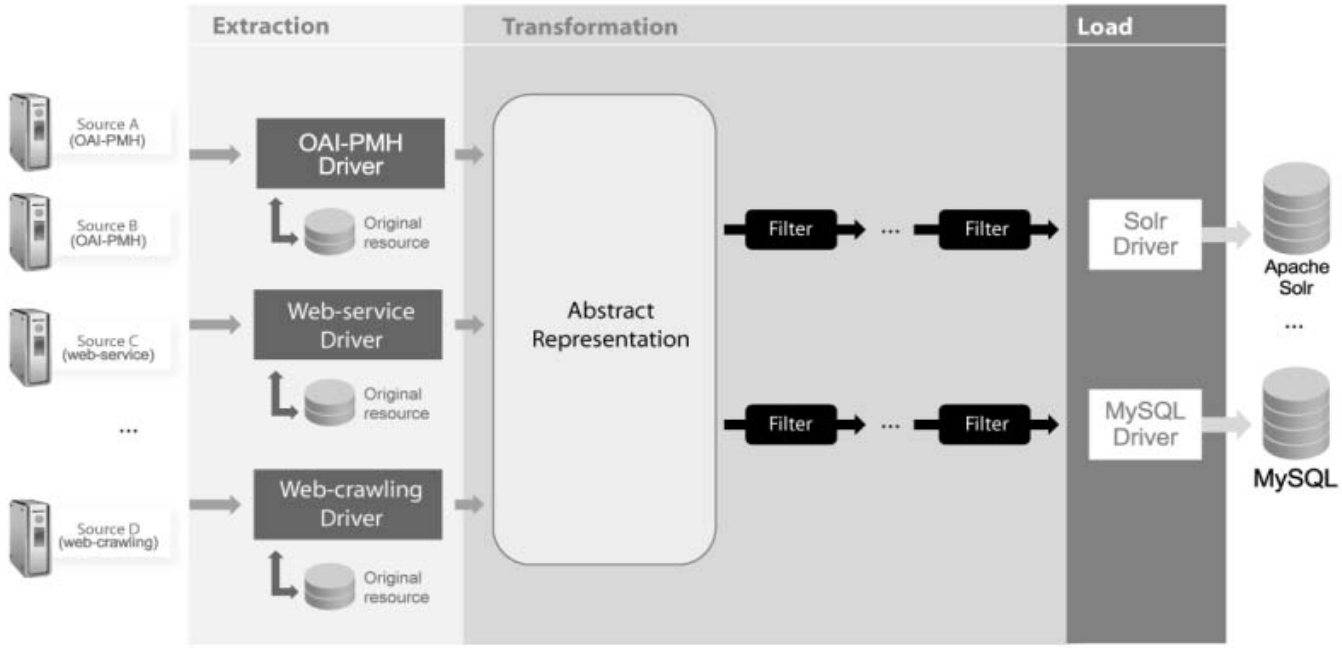

Figure 1. Architecture diagram

DATA MODEL OVERVIEW

This data model is primarily based on three elements, from which the whole model is developed. These elements are Repositories, Harvest Definitions and Collections.

Repositories represent external digital repositories with relevant resources, thus being the object of data collection. A repository is an abstract entity that does not determine how to obtain resources and only registers general information such as the name of the source institution, contact e-mail, Web site, etc. In order to harvest resources from a specific repository, connection drivers –components with the required logic to establish connections– must be first associated, determining the relevant parameters.

A *Harvest Definition* element comprises all the specifications required to carry out a harvest instance (or particular harvest). That is, harvesting processes are performed according to the harvest definitions in the adequate status –i.e. still have jobs to carry out. A harvest definition is created from a connector associated to a repository, thus specifying the protocol or harvest method used. This allows the creation of multiple harvests on a single repository, using different communication approaches.

Collections are the third important element. They represent the various end targets for the information generated after applying transformation and analysis processes to harvested resources. As is the case with repositories, collections are an abstract element within the system, and this means that each collection has an associated connector that determines the storage method and its corresponding parameters. The main goal is to allow the use of the different storage options, not only based on the storage type, but also the type of information to be stored. For example, let us consider a collection that specifies storage into the file system as backup, another collection that specifies insertion into an Apache Solr [7] core for resources identified as Thesis, and a third collection that specifies insertion into another Apache Solr core for resources written in Spanish.

The data model is completed by the three main elements described above, adding elements associated to connectors and to harvest definitions, supplementary information about repositories and additional elements for controlling and tracking harvesting methods.

EXTRACT

Extract is the first phase in resource harvesting, carried out in different stages. The first is the determination of harvest definitions that must be loaded in order to be run. For this purpose, each definition has scheduling information that specifies the date and time of the next execution. Since harvest definitions are the actual extraction jobs, they contain a reference to the interacting connector to establish the connection and download the information. Likewise, harvest definitions are narrowed down, adding supplementary information –usually parameters– about the associated connector protocol. Specifically, the connector is the component that carries the logic required to establish the connection, and the harvest definition specialization contains the particular harvesting parameters.

In some cases, harvesting jobs must be carried out in stages, due to a number of reasons: data volume is too large and thus must be partitioned, organizational issues, etc. An actual case is OAI-PMH protocol, which allows incremental harvesting by date range. This is shown in the data model, when harvest definitions are decomposed into *actual harvests*. For example, an OAI harvest can be created specifying a date range of one year and then split that harvest into one-month separate harvests, which will generate twelve harvests that must be completed to meet the requirements of the initial definition. This also reduces losses due to system crashes, since only the job associated to a part of the harvest would be lost, and not the whole harvest itself.

This fault-tolerance is achieved using *Harvest Attempts*. That is, for each connection a new attempt is registered, and it will remain valid until the harvest is completed or an interruption occurs, either by the administrator, a server timeout, errors in the responses, system crashes, etc. There is a configurable limit that determines the number of attempts to try before disabling the harvest.

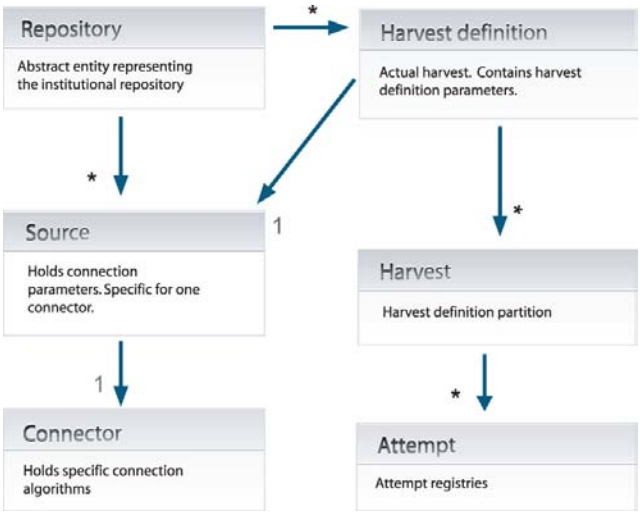

Figure 2. Extraction phase data model abstraction

Downloaded information is handled by a general handler common to all connectors which stores harvested data locally and retrieves them when needed. For example, a particular handler can store data as files on disk.

TRANSFORM

This phase initially transforms the harvested resources to a simple abstract representation that allows uniform processing of all resources. This transformation is done by connectors, since they contain information about the original representation and the rules that must be applied to take it to an abstract level. Each resource, already in their abstract representation, goes through a filter chain in order to analyze particularities and modify data, if necessary. The system comprises a predetermined set of independent filters, which are simple and reusable components that act according to parameters specified in a filter configuration file.

As seen above, each harvest definition refers to a target collection set. Each collection specifies a set of filters that must be applied before inserting a resource into that collection, where selection order determines their application.

Filter execution may lead to modification, adding or erasing specific resource data (metadata values); depending on specific filter functions and configuration.

Available filters on this application include:

- *CopyField*: Copies content from field to field. If the target field is nonexistent, the filter creates one.
- *DefaultValue*: Determines if there is a nonexistent or valueless field. If this is the case, it creates a new one with a predetermined value.
- *FieldRemover*: Takes a field list and removes them from the resource.
- *Tokenizer*: Takes field values and tokenizes them from a specific character series, generating multiple additional values.
- *Stack*: Aggregates filters; defines a filter list (with configuration and order) to ensure the order of application.
- *ISOLanguage*: Applied to a field that specifies the resource language, searches for the field value in a language list and replaces the original value with the ISO-639 language code found.
- *YearExtractor*: Applied to a field that contains a date, extracts the year and saves it on a new field.
- *Vocabulary*: Takes field values and contrasts them against a dictionary, unifying word variations and synonyms into a single word.

LOAD

This is the third part, when transformed resources are sent to data stores, completing the scope of this tool. For this purpose, each

collection refers to a target connector that contains the data store logic required to interact with this latter.

After going through the transformation stage, resources in their abstract representation are sent to the connector associated to the target collection, where they undergo further transformations to produce an adequate representation that matches data store requirements.

MANAGEMENT

Loading of repositories, collection, harvest definitions, filter selection and so forth is managed through a web application. This web application is included in the software and allows management capabilities to handle all aspects that make up the tool. More precisely, it allows management of collections, repositories, harvest definitions, connectors –source and target-, languages, publication types, users, roles, system parameters, collection assignments in a harvest definition, filter selection from a collection, among others. Besides, it has a special section to control the execution of collection and storing, from which these processes can be independently initiated and interrupted, creating a real time report of the jobs that are being run.

Finally, simple reports associated with repositories are created to show the status of completed harvests -number of failed harvests, number of harvests with no register return, etc. - average daily resource downloads, total volume of document downloads, and more. Analogously, resource distribution by source target is shown for each collection, specifying amount and proportion represented by each one in the whole collection.

FUTURE RESEARCH

This tool has a number of features that allow for further improvements or extensions. Key aspects include:

Transformations: The most important extension point seems to be focused on transformations, since they allow the application of interesting processes to the collected information.

Semantic extraction: Detects relations among resources based on the information they contain.

Fulltext download: Identifies an URL pointing to the fulltext and attempts to download the document, to apply further filters to its content.

Author standardization: Analyzes the author's name to generate standardized metadata.

Duplicate detection: Provides techniques to avoid insertion of two resources –probably from different sources- when they represent the same resource.

CONCLUSION

This document discusses a recurring challenge faced by digital repositories that arises from resource collection from diverse sources; then shows how an architecture used mainly in the business area can provide a solution for these issues. The three main ETL architecture phases cover each one of the activities performed during the resource collection work in the context of digital repositories, making it an adequate approach within this area, allowing for further improvements and extensions, as seen above.

REFERENCES

[1] The Open Archives Initiative Protocol for Metadata Harvesting, http://www.openarchives.org/OAI/openarchivesprotocol.html

[2] A. Maedche, M. Ehrig, S. Handschuh, R. Volz, L. Stojanovic, "Ontology-Focused Crawling of Documents and Relational Metadata", Proceedings of the Eleventh International World Wide Web Conference WWW2002, 2002.

[3] P. Minton, D. Steffen, "The Conformed ETL Architecture", DM Review, 2004, http://amberleaf.net/content/pdfs/ConformedETL.pdf.

[4] Data Warehouse, http://en.wikipedia.org/wiki/Data_warehouse.

[5] Data Mart, http://en.wikipedia.org/wiki/Data_mart.

[6] Business Intelligence, http://en.wikipedia.org/wiki/Business_intelligence.

[7] Apache Solr, http://lucene.apache.org/solr.

[8] M. L. Nelson, J. A. Smith, I. Garcia del Campo, H. Van de Sompel, X. Liu, "Efficient, Automatic Web Resource Harvesting", WIDM '06 Proceedings of the 8th annual ACM international workshop on Web information and data management, 2006.

[9] C. B. Baruque, R. N. Melo. "Using data warehousing and data mining techniques for the development of digital libraries for LMS", IADIS International Conference WWW/Internet, 2004.