Batch Loader Requirements Report v1.0

Jethro Shell

November 26, 2012

1 Introduction

In this document, an outline of the requirements will be given for the batch loading process. Its structure will be as follows: A brief outline of the goals of the batch loader will be set out, followed by a discussion of the overall process itself. Based upon the requirements, further detail will be given for accomplishing each goal.

2 Goals

The batch loader is required to perform certain actions. Below are a list of the goals of the application in regard to the overall project requirements. These have been given priority statuses. Those that are of **High** status are deemed to be a minimum requirement set.

High To import data from the Partner Organisations (PO's) to the Fuzzy Photo data warehouse.

High Convert the format of the supplied PO's data to the specified criteria in a robust manner.

High Design and construct a long term maintainable solution.

High Keep to a minimum the need of the PO's to manipulate the data format.

High Maintain the relevance of the data held within the Fuzzy Photo data warehouse through periodic updates.

Medium Produce a generic, dynamic approach to the conversion of differing data types to the required data format (for example, MS Excel to XML).

Medium Produce a generic, dynamic approach to the mapping of client data schemas to the specified Fuzzy Photo data schema (currently LIDO schema).

3 Process Structure

The batch loader is an element of the overall delivery of information within this project. It will act as a preprocessing and maintenance support application to the data held within the warehouse structure. Figure 1 depicts a high level model of the batch loader process.

Within this abstracted view, four primary elements are evident to complete the process.

1. Request and receive data.
2. Convert data format to correct type.
3. Map data schema to correct version.
4. Import data to warehouse.

Taking each of these elements in turn, the requirements and possible options to accomplish them will be presented.

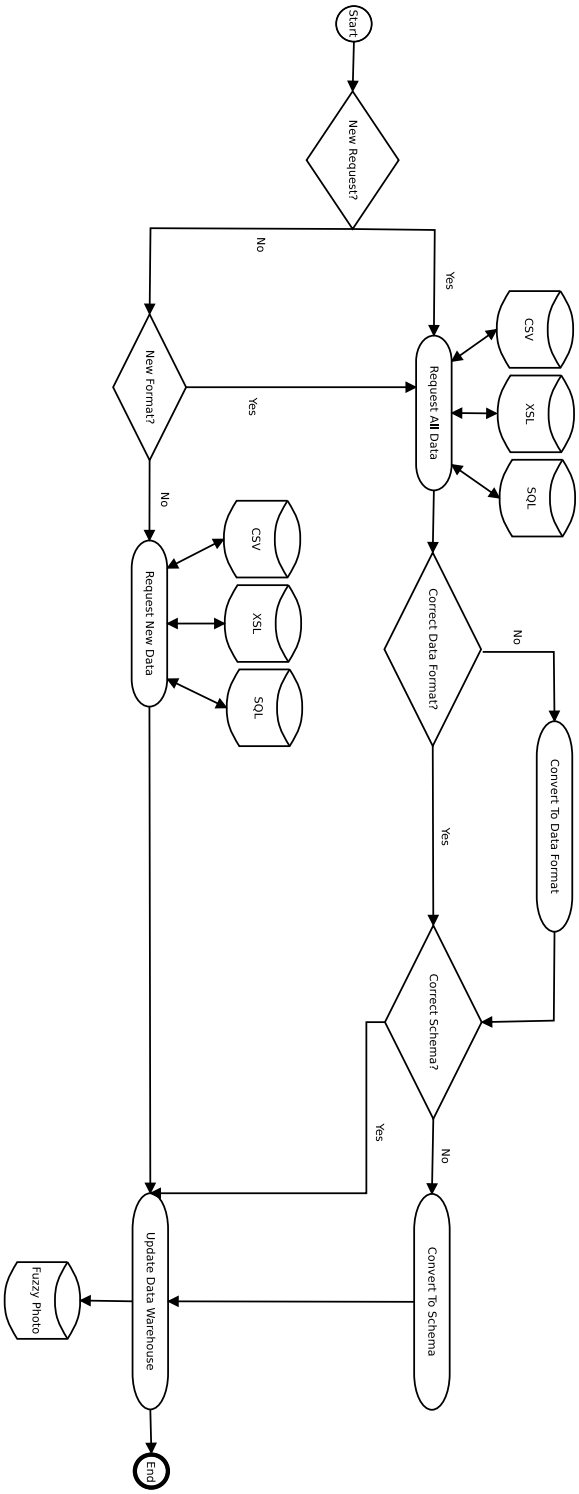

Figure 1: An Outline of the Batch Loader Process.

3.1 Request and Receive Data

3.1.1 Requirement

The application requests data from a source. Dependent on the location, the availability of the source will need to be ascertained. The receiving of the data may require predefined knowledge of the data type. This can be accomplished *a priori* through interaction with the Partner Organisations (PO's) or by the use of other procedures.

3.1.2 Approach

The gathering of data will be heavily dependent on the format of the source from each Partner Organisation (PO). Initial delivery of data may come through varying media. The importing of data to the batch loader, as a result, needs to be sufficiently robust to manage local and distributed file locations.

The locality of the file system to the batch loading application can be absorbed within the network structure. Access to distributed files can be accommodated with the use of secure network traffic or through the use of a web-based interface.

A standard application of a Relational Database Management System (RDMS) such as MySQL along side the use of a web based server allows for the distribution of data. Equally implementations such as SPARQL Protocol and RDF Query Language (SPARQL) allow for the querying, retrieval and manipulating of data from remote locations. Through the connection to a SPARQL endpoint a user can query a knowledge base via the SPARQL language retrieving the required data in Extensible Markup Language (XML) format. Currently a number of museums and archives are using the SPARQL interface including the British Museum Collection.

3.2 Convert Data

3.2.1 Requirement

This section of the application will convert single and multiple files across data structures to a unified final file type in accordance with the required specification. Figure 1 shows an example conversion from a relational data format in the form of an SQL database, to the required configuration. Presently this will take the form of an XML structure.

3.2.2 Approach

One of the key elements of the batch loader process is the production of a unified data format. It is envisaged that the PO's will supply data in varying formats. Once this has been identified, there will be a need to convert the data to a single format. Two approaches that can be adopted are an *individualised* and a *generalised* approach.

Individualised Approach In order to convert each file type, an individual configuration file / process can be constructed. This can either be mapped to a separate conversion application or a individual conversion tool can be produced to carry out the necessary changes. The speed of implementation will be proportional to the quantity of file types that are identified in the preliminary data gathering, conducted with the PO's. This approach can supply a robust initial application and satisfy the minimum

requirement set, however changes to data structures within the PO's would require direct alterations to the corresponding configuration file / process.

Generalised, Modular Approach To facilitate the implementation of a generic, dynamic data conversion method, there can be the use of a generalised, global approach. Two steps are required to achieve this method.

The first step is the recognition of the data types. Differing approaches can be used in file type identification. An initial allocation of allowed file type conversions can be used to increase the speed of implementation and robustness. This however can be at the detriment of flexibility and possible legacy. Future proofing the system would require clear communication with PO's (as previously discussed) to instigate changes to the process.

A generalised approach moves towards the absorption of the conversion process into the application. The identification of the file format would be handled through an automated process. Simplistic methods can be employed to ascertain the parameters of a file. Processes such as the use of the Unix **file** program which is able to supply the type of data contained within the file structure using magic numbers. For example:

```
file test.odt test.odt:  OpenDocument Text
```

Tools have also been developed to capture file information. The UK National Archives have developed the Digital Record Object Identification (DROID) tool. Written in Java and part of the PRONOM project, the DROID tool acts as an automatic file format identifier [2].

TrID offers similar file identification. Using a database of definitions which describe recurring patterns for supported file types, TrID identifies files by their binary signatures. A probability rating is returned against each file based on the signature within the database. The definitions database can be updated so any alterations or newly defined file types can be captured.

The second step is the conversion of the file format into the correct specified type. A mapping can be created between specific file types and conversion techniques. There are a number of tools that can be employed to convert non-XML to the compliant XML structure. csv2xml [1] is a BSD/Linux based tool that is able to convert Comma-Separated Values (CSV) files to XML. JSefa [3] approaches the conversion of CSV files through the use of the Java programming language and the creation of an API. There is support for a number of spreadsheets including OpenOffice and LibreOffice conversion [5] and Microsoft Excel formats [4].

Significant research has been applied to isolating file types [10, 9, 7]. Fong et al. [8] propose a the translation of the schema of a relational database into an XML schema through the use of an Extended Entity Relationship (EER) model. From the EER model a stepwise procedure produces the XML output.

3.3 Mapping of Schema

3.3.1 Requirement

In the transitional process of forming the final file type, the batch loader will form the data structure into the correct schema. The schema will be compliant with the parameters set out within the LIDO schema.

3.3.2 Approach

Two overarching approaches can be used to alter the data structure to the final schema (although variations of either approach are viable).

Individualised Approach A simplistic, individualised approach can be used to construct single configuration files that orchestrate the formation of the appropriate schema. Each file will be assigned to individual PO's. Based on known information sourced through interaction with the PO's, the files will be maintained and updated. The process is illustrated in Figure 2.

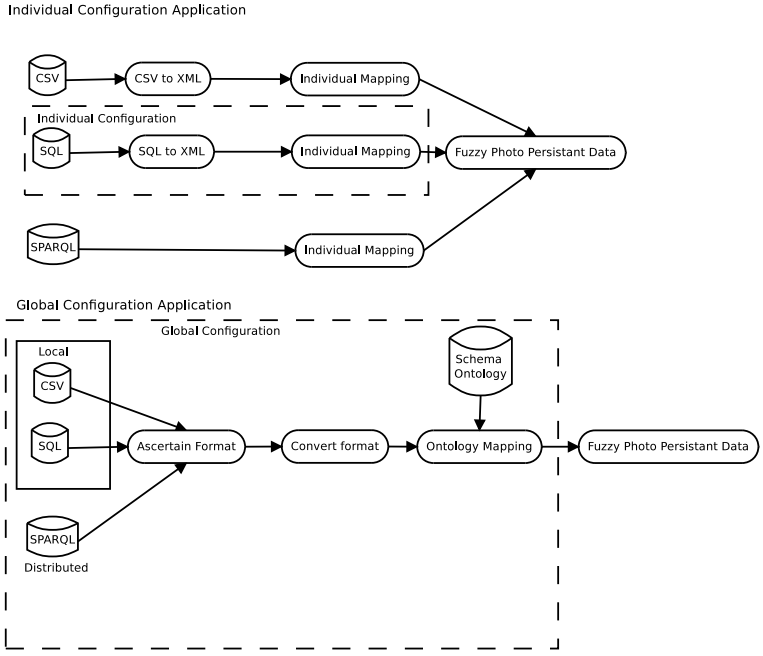

Figure 2: An Outline of the Conversion Process Indicating the Use of Both an Individual and Global Configuration.

The configuration shows the use of individually constructed files that have predefined knowledge of the file format. Based upon this knowledge, the data format of the file is constructed. Similarly, an individual mapping process is used to form the final XML file. This is based on previously sourced information specific to the collection. Individual categories are mapped on an individual basis.

The use of an individualised approach can increase the speed of implementation through a reduction in complexity. There is however, a requirement to maintain strong links with the PO's in order to maintain the persistence of the data held within the system.

Generalised Approach In contrast to the simplified nature of the individualised approach, a global configuration can be used to offer a generalised, dynamic approach. Figure 2 highlights the use of a global configuration. Local and distributed data sources are absorbed into the system through the use of abstraction layer. Based on the file that

is received, a single application determines the file type. Each file type can be either be assigned a conversion process or alternatively a single application can parse all of the required data formats.

Following this initial step, the system maps the desired schema to the data through an integrated procedure. To offer a level of autonomy, a schema ontology can be generated to map the items that are within hierarchically similar elements. Virtuoso's Sponger system implements a similar process [6]. Although focussed on RDF/XML, the Sponger uses a middleware component to offer a pluggable architecture to extract data from one or more sources. Core functionality is offered as cartridges. Ontology mappers map the extracted data to one or more ontologies/schemas on the route to producing Resource Description Framework (RDF) linked data.

Varying degrees of generalisation can be placed within the system. Knowledge elicitation that has occurred through the formation of previous individualised files can form the basis of an ontology so creating a *base of knowledge*. Using this base, semi-automated schema discovery can occur.

The generalised approach allows for a decrease in the use of the *human element* within the maintenance of the batch loader, however this is with the addition of added complexity. Due to the complexity and dynamic nature of the data sources, an element of human post processing will also need to be included to clarify schema mapping.

3.4 Import Data

3.4.1 Requirement

The final data format and schema that are produced are imported into the data warehouse.

3.4.2 Approach

Depending on the basis for the persistent storage strategy, this will be carried out either through the use of an interface with the database or via a connection with the batch loader. A number of database to implementation strategies are possible. These can be summarised in three broad categories:

Relational

- Transactional processing using SQL as an access language.
  - Data is stored on disk in a table structure.
  - Concurrency is controlled through dynamic locking.
  - Mature and widely adopted.

Analytical

- Referred to as Online Analytical Processing (OLAP)
  - Can be referred to as Multiple Online Analytical Processing (MOLAP) and Relational Online Analytical Processing (ROLAP).
  - MOLAP uses multidimensional database for querying, ROLAP uses a relational database.
  - Used for large data stores such as data warehouses.
  - Use of SQL and MDX languages.

NOSQL

- Describes as *Not Only SQL*
  - Non relational and schema free so can cope well with volatile data.
  - Can be distributed, highly scalable with high performance.
  - Able to use disk and memory based access.
  - Multiple structures:
    - Key Value Stores
    - Column Family
    - Document Stores
    - Graph Databases

  Multiple database applications are available for each category. More information regarding the data warehouse structure will be given in a subsequent document.

References

[1] csv2xml - a csv to xml converter. http://csv2xml.sourceforge.net/, November 2012. Online: accessed 23rd November 2012.

[2] Digital record object identification. http://droid.sourceforge.net/, November 2012. Online: accessed 8th November 2012.

[3] Java simple exchange format api. http://jsefa.sourceforge.net/index.html, November 2012. Online: accessed 20th November 2012.

[4] Microsoft office: Export xml data. http://office.microsoft.com/en-gb/excel-help/export-xml-data-HP010206401.aspx, November 2012. Online: accessed 23rd November 2012.

[5] oooexport. http://digitalimprint.com/misc/oooexport/, November 2012. Online: accessed 20th December 2012.

[6] Virtuoso sponger. http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VirtSponger, November 2012. Online: accessed 23rd November 2012.

[7] I. Ahmed, K.S. Lhee, H.J. Shin, and M.P. Hong. Fast content-based file type identification. *Advances in Digital Forensics VII*, pages 65–75, 2011.

[8] J. Fong, F. Pang, and C. Bloor. Converting relational database into xml document. In *Database and Expert Systems Applications, 2001. Proceedings. 12th International Workshop on*, pages 61–65. IEEE, 2001.

[9] R.M. Harris. Using artificial neural networks for forensic file type identification. *Master's Thesis, Purdue University*, 2007.

[10] M. McDaniel and M.H. Heydari. Content based file type detection algorithms. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, page 10 pp., jan. 2003.