EAD, Primo and a data management hub

Lukas Koster
Library of the University of Amsterdam
IGeLU 2016
Developers Day
The problem is the heterogeneous and complicated information infrastructure of the library. Huge amounts of time and effort are spent on moving data from one system/database to another and shoehorning one record format into the next, only to fulfill the necessary everyday services of the university library. This creates critical system interdependencies, data redundancies and proprietary data transformation configurations. See my earlier ELAG 2015 presentation “DataMazed”.
ETL: Extract Transform Load

Transformation is mostly done with utilities that belong to a specific system/tool.

Proprietary transformation and storage

Multiple data formats and transformation tools

The process of getting data from one system/database to another is referred to as ETL: Extract - Transform - Load. All three parts are usually executed with proprietary, system-specific tools and utilities. ETL is the most time and resource consuming portion of the total data infrastructure management organisation.
One result of our Dataflow Inventory project was the DataMap repository, in which we describe all systems and (virtual) databases and the dataflows between them. Including the data formats, data units and transformations. This is only a small part, diagram of a subset of this DataMap repository. DataMap makes very visible, that there are numerous individual, parallel, partly overlapping system and/or data format dependent ETL processes.

“Not only is it not possible to invest time and effort productively in innovative developments, but this fragmented system and data infrastructure is also completely unsuitable for fundamental innovation. Moreover, information provided by current end user services is fragmented as well. Systems are holding data hostage.”

Dependency on systems provided by one or a few companies is sometimes referred to as “vendor lock-in”, something that we need to avoid, we are told. In reality however, there is not much else. We are always dependent on one or more system and database providers (not only commercial ‘vendors’, but also local developers and free and open source software communities). Better to speak of “systems lock-in” (also referred to as “silos”). Anyway, from a system management and efficiency perspective ‘vendor lock-in’ appears to be better than the chaotic multi system and silo environment. This does not mean that you won’t have any problems, but you don’t have to solve them.
yourself. From the innovation perspective however, this is not the case. But in my view there is not much difference here with a fragmented infrastructure.

It would however be great if we can free the data, in such a way as to minimalize (not eliminate) these dependencies, which would also lead to more efficient and innovative investment of people, expertise and funding.
One of the possible alternatives, which we are testing now in a separate pilot project, is a central Data Hub, with some kind of universal and open data format. The idea is that all transformations are done using one universal tool, and all resulting data is stored and linked in one universal data format. The Extract and Load procedures may partly still be dependent on the source and target systems/databases/services.

One of the advantages would be that it will be a lot easier to migrate systems.

Not only existing legacy systems can then get the data from this hub, but also many new services.

A far-away future might be that libraries will use only one central data hub with sources and targets plugged in.
Theoretically, the advantages of such a universal Data Hub are:

- It is independent of systems/vendors (except of course of the Data Hub itself....)
- You only need knowledge of and expertise in one transformation tool
- Migration of systems would be easier because the data transformations will remain available, without having to redo these in the new systems
- On the long run: it should be possible to store all your data of all your content in one place, once, in a reusable format.
As mentioned before, the requirements of a central Data Hub are:

- Central linked storage
- Universal data format
- Data transformation routines
- Support for import and export (in many ways) of common and proprietary data formats
- Last but not least: scheduling all these ETL tasks in batch: no need for manual fiddling
We started a pilot project, as mentioned. Name: CORE (because everybody else already uses “HUB”, like our new library file sharing environment based on SharePoint). CORE can mean anything, whatever you like.

Focusing on a real life practical challenge: getting EAD formatted collection/archives data into Primo PNX, preferably linked to related information from our Aleph ILS.

The problem is: we’re still in the preliminary phase, testing tools etc. Also our collections system/database environment is changing. So there is not much to show. Only a description of the context, the issues and the possible routes.
EAD - Encoded Archival Description

Hybrid format:
- Collection/Archive level
- Nested sub-levels - Items

Focused on Describing, Displaying Collection structure
Variations within standard structure (MARC!)
EAD2 -> EAD3 Differences

EAD is just like MARC a proprietary data format, targeted at just one type of content and just one type of display/end user environment. Both are not suited for generic discovery tools like Primo. And here we have the core problem of our current library information infrastructure: Introverted systems. Legacy workflows.

Just like MARC, EAD is a display format that is incorrectly used as a storage format.

Actually, EAD does not describe one type of content (although it does for archivists) it is a hybrid format: both a single collection level in one EAD file, and multiple multi-level containers and objects, that make up the collection.

Just like with MARC (AACR2, RDA), you can have cataloguing rules, for instance ISAD(G). And just like with MARC every user can implement their own interpretation.

To make the situation complete, there is currently a major upgrade from EAD2 to EAD3, with a number of substantial differences.
Besides the official LoC EAD pages https://www.loc.gov/ead/ I find this EADIVA site very helpful, maintained by Ruth Kitchin Tillman, of Notre Dame Hesburgh Library.

EAD is written in XML, a hierarchical data format structure

Just like MARC it has a Control section, and then a Content section with separate sections for the Collection, and the Containers/Objects
An example of an actual University of Amsterdam Collection EAD.

Some remarks:

- relatedencoding="dublin core", meaning “If only the <control> element (not the entire finding aid) is going to be mapped to a particular standard such as MARC or Dublin Code, this is where one would specify it.”
- <descrules>ISAD(G)</descrules>: enumerates any “rules, standards, conventions, and protocols used in preparing the description.” (Deprecated in EAD3, now: “Use <conventiondeclaration> or <localtypedeclaration> for more controlled forms of citing conventions used in writing the description.”)
The same data, now in the (home grown) collection system display. In Dutch obviously.
The left side is based on the `<eadheader>`/<filedesc> and the `<archdesc>`/<did> etc. parts of the EAD file.
The right side (ToC) is based on the `<archdesc>` and multi-level container parts.
An example of the multi-level EAD section:
<dsc> and nested <c0> - <c12> levels.
If the value of the "level" argument = "file" it describes an actual object.
The <daoloc href= argument contains not a full link/URL/URI, but only the identifier part. The full link has to be constructed with some base-url.
Inventaris van het archief van Hugo de Vries 1866-1935


Collectieonderdelen

1. Persoonlijk leven

1.1. Overig

1.2. 1907, 1929, 1934 en z.d., 2 omstagen


The EAD data from the multi-level section, first top level: in the collection system display.
In current Primo there is basically a flat record structure, describing individual items. There is a very simple Whole/Parts implementation (parts of a serial) with linked internal ID's.

Getting the collection level into Primo is relatively easy. But that doesn't apply to the multi-level structure.

It is not possible to use Primo Normalization Rules to traverse all XML levels, as far as I know. Most libraries that harvest EAD objects into Primo first flatten the EAD-files to some other format (MODS, etc.).
Example of harvested Collection level EAD displayed in Primo.
Note again the <lb/> effect.
Example of the EAD file collection level in Primo PNX.
See the <lb/> tag that is ignored by Primo Normalization and can’t be referenced. So it is just substituted by a NULL.
New in Primo: hierarchical Collection Discovery. Currently only for collection data from Alma and Rosetta. Soon also for third party source systems. All items are available as individual PNX records, with a new PNX link field that links to the higher level object. An additional Primo area ("Lobby") is available for displaying the Collections as units.
Primo Collection Discovery still uses the one-dimensional data structure in PNX, but adds display hierarchy using on the fly API calls to the source system.
Implementation of collection hierarchy in PNX.
Field “Collection Discovery”: control/colldiscovery.
Holds Type of record ($$T$$), ID (collection or parent collection; $$D$$), Institution ($$I$$).
DataHub Requirements

Central linked storage
Universal data format
Data transformation routines
Support for import/export common data formats
Scheduled batch processing

*For EAD/Collections in Primo:*
API: hierarchy structure

To come back to our initial DataHub requirements: for our specific Primo Collection hierarchy need we have to add the option of getting the hierarchy structure through an API call.
Now, that was the context and background. Next step was: looking for and selecting tools for implementing a Data Hub. There are a number of tools and projects out there. Some of them dedicated for a specific project, like DPLA, DigitalNZ. Probably also Europeana has a tool.
We decided on looking closer at d:swarm and catmandu, because these tools are open source, not dedicated for one specific project, and relatively easy to use.
d:swarm is developed by SLUB (Saxon State and University Library) in Dresden, Germany, together with Avantgarde Labs).

Open Source.

It theoretically meets all the requirements for the data hub. There are two versions:

- Streaming: no data storage, just ETL
- DataHub: with data storage
DataHub store: Neo4j Graph database

ETL: Graphical UI (Metafacture)

TPU/Backend: Task Processing Unit - batch/scheduling...

API

Ne04J Graph database is used as the DataHub storage, where all imported data is transformed to graph relations, and linked where obvious.

For Mapping and Transformation of data the graphical UI Metafacture tool is used.

It has TPU (Task Processing Unit) for batch processing of ETL procedures. And there is an API.
The d:swarm team was so kind to provide us with a temporary virtual installation on their servers for free, for evaluating purposes.

Mapping of data from source to target is done in the graphical UI. EAD can be handled as XML, with manual mapping.
Data transformation is done in the graphical UI, with adding and specifying routines, much like primo normalization rules.
Many common import and export data formats are available, although not EAD. But there is Primo PNX.
Example of results of mapping the EAD collection level to PNX. No object level yet (hierarchical/conditional mapping is not available as such).
Catmandu, developed by University Libraries of Gent, Lund, Bielefeld. Open Source.
Fully command line operated.
Consists of many optional modules.
Many common import and export data formats, although not EAD and Primo PNX.
Storage is available via a number of tools.
Data transformations are performed using “fixes”, small modules that perform a single operation.
It is really easy to install Catmandu core and additional modules, using CPAN.

Again, EAD is imported in catmandu with the XML Importer.
Data transformation from one format to another is easy using the available Exporters. Here an example with JSON output.
Also storing data is easy with the Store modules. In this case XML to MongoDB, and then exporting from MongoDB to JSON.
The same example with YAML output.
Catamandu - Fixes

Data transformation: Fixes (functions)

*Such as:*

- `split_field(author, "")`
- `marc_map('245', title)`
- `remove_field(isbn)`
- `substring()`
- `viaf_read('number')`
- `xml_simple()`
- `xml_transform()`

In Catmandu, data transformations are done with available Fixes, or create your own.
An example of data transformations: using the marc_map fix to map input fields to MARC tags, and keep just one field using the retain_field fix.
CORE Project: Issues

Still in exploratory phase: tools evaluation

Good support from both tools’ developers
- d:swarm - Thomas Gängler and colleagues
- Catmandu - Patrick Hochstenbach and friends

Our CORe project issues: we still haven’t started the actual pilot. Still in preliminary phase of evaluating and selecting tools.

Both d:swarm and Catmandu have many dedicated developers that are very willing to give support.
The d:swarm graphical mapping tool currently has some performance and display issues. This makes it hard to work with. There is no EAD importer (yet). The d:swarm team is of course willing to develop one, given the time etc. The hybrid hierarchical structure makes this not an easy task.

We haven’t had the time to test the batch processing yet.
CORE Project: Issues: Catmandu

Command line: fast!

Needs a lot of framework, scripting, etc.

EAD: not supported (yet)

PNX: not supported (yet)

Unlike the d:swarm graphical interface, the Catmandu command line is fast. However you have to be very proficient in UNIX/Linux command line work to be able to use it correctly and efficiently. Of course for an actual DataHub this means putting together a vast and complex framework of prepared scripts and batch processes. Which will take a lot of work and resources to set up.

Neither EAD nor PNX is supported (yet), but the Catmandu community is willing to develop importers and exporters for these data formats.
CORE Project: Issues - EAD

EAD hybrid multilevel structure: problematic

EAD is a presentation format (like MARC)

Amsterdam University Library internal project: Assessing EAD workflows, starting September 2016

EAD is a data format has a number of issues too. We have already named most of them. Most importantly: the hybris multilevel structure and the use of a display format as storage and exchange format.
CORE Project: Issues - Primo

Collection Discovery for non-Ex Libris collection management tools not available yet

Primo Collection Discovery is not yet available for third party collection/archive systems, so we can’t test that at the moment.