Technical report - version 1.0
Author: Matthias Palmér
Date: 2014-08-29

Bilaga 2: MetaSolutions view on a linked data architecture for GLAM - applied to the Swedish Open Cultural Heritage project K-Samsök

In this report we try to identify a range of functions that are useful when aiming for a working LOD ecosystem for the GLAM domain (i.e. Galleries, Libraries, Archives and Museums). The assumption is that there is a range of data providers that produce data and another organization is responsible to act as an aggregator, that is providing added value by combining the data. The added value can be both richer search results and views of the data as well as making it easier for third-party developers to provide new services that makes use of the data.
The report identifies a range of different functions that are useful in a perspective of collecting and combining data from different sources. There is also a discussion of which functions that fit well together, which to avoid and how this compares to the current and future SOCH (Swedish Open Cultural Heritage project, K-Samsök in Swedish).

About this report

This report is written based on experience from a range of projects and research within the area performed by the founders of MetaSolutions. However, the authors would like to make the reader aware that the report has not, at this time, been thoroughly grounded in literature review or pragmatic tests as the authors would prefer if time would allow.

Linked Data infrastructure functions

The following functions provide complementary solutions in an environment where the goal is to provide access to linked data from different origins and different service levels.
Notice that some functions can be deployed on their own while others needs to be combined to make sense.

Store function

Provides a solution for storing linked data, that is, a triplestore. Preferably the store should provide a capable API to allow services and clients to be built on top of it. From a read-only perspective the API should coincide with the principles of linked data. From a read-and-write perspective the principles outlined by the linked data platform working group is a good starting point.
A good store functionality should also provide fine grained access control to allow for various forms of collaboration and work division when maintaining the data.

Import function

The import function provides a one-time imports of data from a range of sources. This functionality is often very specific or even corresponds to a manual task of collecting data at some point in time and making it available as input to a store, cache or the mapper function.

Harvester function

The harvester function is responsible for collecting data from a range of sources. Information regarding when the data was retrieved and knowledge of for how long the data is considered up-to-date should be maintained in a source independent format. Optimally, the harvester should be capable of understanding several different formats such as OAI-PMH in addition to pure linked data.
In general, the harvester needs to work closely with the cache component and, if the sources are not natively linked data, also with mapper functions.

Minting function

The minting function is responsible for creating new URIs according to various schemes. The different minting schemes may be triggered for specific sources or parameters. Although a single URI minting function is expected to generate URIs with a common base. Typically the URI minting is used by Mapper and Co-reference functions.

Rebase function

The rebase function is responsible for transferring RDF expressions centered on one URI to another URI, typically one with a domain under control of the organization setting up the function.

Mapping function

The mapping function is responsible for transforming various formats into RDF and linked data. The mapper function should be indifferent to the protocol used to retrieve data (this is the responsibility of the harvester function). This function relies on either the minting or the rebasing component to provide URIs for those things that lack a web-friendly identifier or lack an identifier altogether.

Co-references function

Provides a mechanism to identify data that talks about the same thing using different URIs and provides sameAs references to connects them together. There are two reasonable approaches, one is to make sure there are sameAs references from a central URI that are provided by a minting function. The other approach is to make sure all URIs refer to all other URIs representing the same thing via sameAs. The first approach is preferable as the latter both modifies the original data and potentially leads to a much larger amount of sameAs links.
To discover which URIs correspond to the same thing inverse functional properties or making use of the transitivity of sameAs can be used. If that is not enough some form of configurable heuristics for example similar names, titles, or more advanced graph similarity can be used to

make guesses about which URIs that correspond to the same thing. In each case, the rationale for each created co-reference should be provided as metadata for later inspection, manual adjustment, confirmation, or rejection.

Cache function

The cache function is responsible for providing reliable access to data from different sources together with provenance information. Reliable in this context means that you can be sure to get a response or a suitable HTTP error message within a certain time frame. If the amount of data is too large it should be paginated to be more useful for for small clients. The cache also simplifies access when there are 303 redirects, reliability issues with certain providers as well as provides content-negotiation for a wider range of RDF formats that not all providers support.
The cache function may also includes a SPARQL endpoint.

SPARQL search function

This is the most obvious function, simply provides a good SPARQL endpoint.

Field search function

A SPARQL endpoint is often very useful, but in some situations a more dedicated text-search approach is needed. A common approach is to use Solr or similar technologies to index the linked data expression and provide an API where all or a few of these values can be searched.

Materialization function

The materialization function is responsible for keeping a set of inferred statements based on RDFS, OWL or various rules. Examples include explicit type information to all superclasses, and all statements implicit from subproperty relations. The exact amount of statements to materialize should be configurable and depend on the needs and capabilities of the expected clients. The materialization function is dependant on the cache function.

Enrichment function

The enrichment function provides a mechanism to enrich existing data in a store or cache. The enrichment information can originate in manual input, e.g. in the form of translations, categorizations, tags, comments etc. Enrichments can also be the result of automated processes, e.g. NLP based information extraction. An example in the cultural heritage domain is to automatically detect and connect cultural heritage objects to their digital representations, for example, connect a church with photos of the same church.
An important aspect of the enrichment function is that the enrichment data should be provided as linked data and has a clear relation to the original source data.

Look-ahead function

The look ahead function provides specific information about outgoing links for a thing in a single request. Typically when you request information about a thing, outgoing links are represented by a simple triple, giving you information about the link type (predicate) and the URI (object) of

the thing linked to. In most cases you need just a little more information of the thing linked to be able to decide if you need to consider that thing or not in your scenario. For instance, the type and label is often sufficient. However, in general, to get that information means that you have to make another request. The look ahead function builds upon the cache function to allow you to specify which information of the linked to things you want to have included in the response.

Export function

The export function resembles the look-ahead function as it provides a way to collect information about several things in a single request or a larger data dump. The set of information to export could be determined by a manual list of things, a subset dynamically generated from a query or a dump of all information available for the repository / dataset.

Combinations of functions

The functions above can be combined in different ways to achieve different results. Let's focus on four hypothetical combinations of functions for an organization that wants to accomplish something like SOCH, i.e. to provide good access to linked data originating from a range of different sources. Hence, we do not consider the perspective of the provider himself in this discussion. We will first consider four combinations of functions, we refer to them as *naive*, *takeover*, *non-attribution*, *pragmatic* and *complete*. Then we make a short observation of the suitability of certain functions. Finally we consider the current and future implementations of SOCH.

Naive - harvester + cache + SPARQL

In a perfect world every source is provided as linked data and all you need to do is to harvest and provide it through a cache to allow easy access and SPARQL queries that can span multiple sources efficiently. This approach relies on data sources that have already provided links to each other and that there are few duplicates. That is, every thing should have a natural master record maintained by a single organization under a well known URI that can be linked to instead of duplicating information.

Takeover - import + mapper + minter + store + SPARQL

In a sentence, this approach provides a one time import and mapping of data into a triplestore with new URIs accessible as linked data and searchable via SPARQL. Since there is no regular harvesting and the data is provided under new (minted) URIs the original information is hidden in the process, hence the name takeover. It can be argued that in situations when most data-sources are provided in non-linked data formats there is little alternative, however, switching import to harvester and considering a cache rather than a store significantly improves the feeling of this approach.

Non-attribution - import + rebaser + store

This scenario is very similar to the takeover, the only difference is that the data-sources are already available as linked data and still the existing URIs are rebased into own URIs. Although

it is considered bad behaviour to omit attribution, the intention can still be good, e.g. to make data available in a reliable manner which is consistent with linked data principles.

Pragmatic- harvester + mapper + minter + cache + SPARQL + export

Comparing to the naive approach the first thing to realize is that not every source is already available as a linked data source. This prompts the inclusion of a mapper and a minting function to provide URIs for those things that lack or have non-web friendly identifiers. This is a better option than the takeover and stealing combinations above.
In addition, it is expected of a linked data provider today to also provide a SPARQL endpoint. Providing dumps of the data, that is, the export function, is a nice feature that the authors believe is a good practise that should be the norm to provide.

Complete - harvester + mapper + minter + co-references + enrichment + materialization + cache + look ahead + export + SPARQL + field search

The next problem to solve is how to handle duplicates, that is, when the same thing appear in multiple sources with different identifiers. The co-reference component handles this by introducing new identifiers and maintaining sameAs links between the URIs that correspond to the same thing. In parallel to the co-references, additional linked data can be provided automatically or semi-automatically by the enrichment function by using NLP or other techniques.
Second, with highly interlinked data a simple cache is not enough. When you build clients you need to avoid making too many requests to get the relevant data, that is, you need the look ahead function. Third, some clients find it easier to download dumps, especially if they need offline support, i.e. the export function is useful here. Fourth, many clients are simple and cannot make the necessary inferences themselves, making implicit information like class and property hierarchies explicit can be much valued and is provided by the materialization function. Finally a SPARQL search is a good thing, but having a text based search against specific fields in parallel is even better.

Functions to avoid

From the takeover and non-attribution combinations above it can be argued that some functions should be avoided by aggregators:

Function	Reason for avoiding it
Import	One time imports is bad since the data goes stale, use a harvesting function instead.
Rebase	Never rename existing URIs, use a minting function instead to create new URIs that refer back to existing if needed.
Store	Never store data from another source permanently as it were your own,

	use the cache function instead. (Note, if you are a data owner you need to have a store, also, some of the functions like enrichment and co-references needs something like a store internally to work properly, but this is not for the main data.)

Analyzing the current SOCH

The current SOCH uses the following functions:

Function	**How it is used**
Harvester	SOCH harvests via OAI-PMH from a range of organizations within the cultural heritage area.
Mapper	Provided software simplifies the conversion from established databases solutions.
Minting	SOCH provides new URIs for every metadata record harvested.
Cache	All harvested data is stored in a read-only database solution.
Field search	A solr based search API is provided.

How a future SOCH could look like

If the goal is to be a good linked data citizen it is reasonable to aim for the pragmatic combination outlined above. Comparing with the current SOCH only the SPARQL and export functions are missing together with also supporting harvesting from pure linked data sources. However, if the aim is to push the uptake and use of linked data within the cultural heritage domain, perhaps a more ambitious goal should be set that are closer to the complete combination outlined above. The most interesting option is to implement a co-reference function as it is useful in both helping data providers to produce more capable data as well as give consumers of the service a better view of the data. It is also useful in a more competent delivery to Europeana, making better use of the EDM where entities are tied together via the proxy mechanism (can be driven via sameAs links). The enrichment function is also of great interest as it can be of use to make the connection between digital objects such as photos with corresponding physical or historical objects and events.
If the aim is to encourage app developers to make use of the service, it is recommended to consider also supporting the look-ahead and materialization functions as they greatly simplify the task of using the data, especially when the data becomes more interlinked and complex.

Below is an illustration of SOCH's current and future support for the functions. Green color indicates current support, red color indicates the pragmatic option while orange indicates the more ambitious but also more rewarding and complete option.