D1.3.1 – Semantic workflow tool available

This deliverable describes the software to automate the workflow of constructing semantic representation of collection objects and establishing links between vocabulary concepts.

ECP-2008-DILI-528001

EuropeanaConnect

D1.3.1 – Semantic workflow tool available

Deliverable number/name	*D1.3.1*
Dissemination level	*Public*
Delivery date	*9 October 2011*
Status	*v.1.0*
Author(s)	*Jan Wielemaker, VUA* *Victor de Boer, VUA* *Antoine Isaac, VUA* *Jacco van Ossenbruggen, VUA* *Michiel Hildebrand, VUA* *Guus Schreiber, VUA* *Steffen Hennicke, VUA*

e*Content*plus

This project is funded under the *e*Content*plus* programme,
a multiannual Community programme to make digital content in Europe more accessible, usable and exploitable.

Österreichische Nationalbibliothek EuropeanaConnect is coordinated by the Austrian National Library

Distribution

Version	Date of sending	Name	Role in project
0.1	10.10.2011	Luca Dini Ralf Stockmann	Task 2.4 Lead Task 3.3 Lead
0.2	10.10.2011	Jan Wielemaker	Task 1.3
0.3	14.10.2011	Jan Wielemaker	Task 1.3
0.4	20.10.2011	Veronika Prändl-Zika	PM
0.5	25.10.2011	Max Kaiser	PC
1.0	28.10.2011	EC, EuropeanaConnect Web site, liferay	

Approval

Version	Date of approval	Name	Role in project
0.2	10.10.2011	Luca Dini	Task 2.4 Lead
0.3	14.10.2011	Ralf Stockmann	Task 3.3 Lead
0.5	27.10.2011	Max Kaiser	

Revisions

Version	Status	Author	Date	Changes
0.1	Draft	Jan Wielemaker, Victor de Boer, Antoine Isaac, Jacco van Ossenbruggen, Michiel Hildebrand, Guus Schreiber, Steffen Hennicke	09.10.2011	Initial version
0.2	Draft	Luca Dini	10.10.2011	Some comments
0.3	Draft	Ralf Stockmann	14.10.2011	Some comments
0.4	Draft	Jan Wielemaker	20.10.2011	Changes according ot reveiwers comments
0.5	Draft	Veronika Prändl-Zika	25.10.2011	Layout and editing
1.0	Final	Veronika Prändl-Zika	28.10.2011	Finalisation

Table of Contents

1 Introduction

In order to convert collection metadata to the semantic representation as defined by the Europeana Data Model, a semi-automatic workflow is needed. The input for this workflow is the original collection metadata as provided by aggregators or individual cultural heritage institutions. The result of the workflow process is the collection metadata in semantic format (RDF) as represented in the Europeana Data Model. Links are established between vocabulary terms used in the collection.

1.1 Workflow overview

Figure 1 shows the general workflow for the conversion and linking of the provided metadata. Both components are packages of the ClioPatria semantic web toolkit that is described below. For this workflow, we assume that we are provided with an XML representation of the metadata. This can be a separate file or the result of OAI harvesting.

In the first step, the XML is converted to crude RDF format. This is done using the XMLRDF tool, which is documented in section 2. This crude RDF is then rewritten in EDM-compliant RDF, which is done using graph rewrite rules which are executed by the XMLRDF tool.

In the third step, links are established between vocabulary concepts that are used in the collection metadata and Europeana and other vocabularies. For this, we use the ClioPatria-based AMALGAME tool, which is documented in section 3.

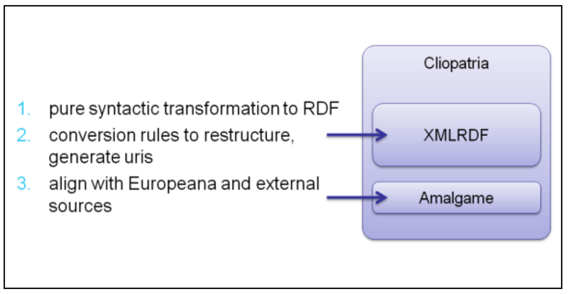

1. pure syntactic transformation to RDF
2. conversion rules to restructure, generate uris
3. align with Europeana and external sources
Cliopatria
XMLRDF
Amalgame

Figure 1: General workflow for converting and linking metadata

1.2 ClioPatria

All workflow components are packages for the ClioPatria semantic web platform. (http://cliopatria.swi-prolog.org/) ClioPatria provides a common ground for the tools through the following functions:

1. A web-interface for inspecting (intermediate) results. This web-interface is used in the initial conversion process to decide on which additional conversions rules are needed to arrive at an appropriate semantic representations that is compliant with the Europeana (EDM) specification.

2. A package management system that can handle versioning and dependency tracking. This allows us to distribute the described tools in a modular fashion.

ClioPatria runs on top of SWI-Prolog (http://www.swi-prolog.org/). SWI-Prolog runs on Windows (all versions later than NT4, both 32- and 64-bits), MacOS X and almost all Unix/Linux versions. ClioPatria and the packages described in this deliverable are distributed though GIT(http://git-scm.com/). GIT provides, in addition to a download service, versioning and digital signatures to ensure integrity of packages.

Download and installation instructions can be found on the respective pages:

- http://www.swi-prolog.org/Download.html
- http://git-scm.com/download
- http://cliopatria.swi-prolog.org/help/Download.html

2 XMLRDF documentation

Home page: http://semanticweb.cs.vu.nl/xmlrdf/

2.1 Introduction

Many datasets are transferred as XML, providing a tree-based datamodel that is purely syntactic in nature. Semantic processing is standardised around RDF, which provides a graph-based model. In the transformation process we must identify syntactic artifacts such as meaningless ordering in the XML data, lacking structure (e.g., the *creator* of an artwork is not a literal string but a person identified by a resource with properties) and overly structured data (e.g. the dimension of an object is a property of the object, not of some placeholder that combines physical properties of the object). These syntactic artifacts must be translated into a proper semantic model where objects and properties are typed and semantically related to the EDM, which is based on the widely used SKOS and Dublic Core vocabularies.

This document describes our toolkit for supporting this transformation process, together with examples taken from actual translations. The toolkit is implemented as a package for ClioPatria. The homepage of this package is http://cliopatria.swi-prolog.org/packs/xmlrdf. The name of the package is **xmlrdf** and the package can thus be installed from ClioPatria using this command:

?- **cpack_install(xmlrdf).**

Deploying the package typically involves the steps below. The commands for managing and running the conversion and rewrite rules are described in section 2.7.

1. Copy the script files from an example.
2. Adjusting the locations of XML source-file(s) and tailor the initial conversion using options to the conversion process and/or a partial RDF schema file. Unless the data is closely related to the example, the graph rewrite rules are normally removed from the rules file.
3. Run the XML/RDF conversion. Connect your browser to the web-address that is displayed when ClioPatria is started and examine the data. Typically, this starts by visiting the Graphs page and after selecting the data graph, selecting the predicates link. This shows the defined RDF predicates with a distribution of the associated *Subjects* and *Objects*.
4. Add some rules to the rules file and run them. Iterate over step 3 and 4 until the desired result is reached.

The remainder of this document describes the details of the conversion process and the used rule-language. This document does not assume any knowledge about Prolog. Deploying the RDFXML toolkit typically requires very little knowledge about Prolog. The specification of the initial conversion is normally limited to changing file-names and passing additional parameters to the conversion process. The rule-language is, as far as possible, a clean declarative graph-rewrite language. The transformation process for actual data however can be complicated. For these cases the rule-system allow mixing rules with arbitrary Prolog code, providing an unconstrained transformation system. We provide a (dynamically extended) library of Prolog routines for typical conversion tasks. In some cases these will not satisfy, in which case expertise in programming Prolog becomes necessary.

2.2 Converting XML into RDF

The core idea behind converting ‘data-xml' into RDF is that every complex XML element maps to a resource (often a *bnode*) and every atomic attribute maps to an attribute of this bnode. Such a translation gives a valid RDF document, which can be processed using graph-rewrite rules.

There are a few places where we must be more subtle in the initial conversion. First, the XML reserved attributes:

- The **xml:lang** attribute is kept around and if we create an RDF literal, it is used to create a literal in the current language.
- **xmlns** declarations are ignored (they make the XML name space declarations available to the application, but the namespaces are already processed by the XML parser).

Second, we may wish to map some of our properties into rdfs:XMLLiteral or RDF dataTypes. In particular the first *must* be done in the first pass to avoid all the complexities of turning the RDF back into XML (think of the above mentioned declarations, but ordering requirements can make this fundamentally impossible).

If the source contains data that we want to represent as XMLLiteral, we must provide information on how to identify the XML we want to preserve. This is achieved by providing a (partial) target RDF Schema file, where predicates and classes are related to the XML source using an proprietary predicate called **map:xmlname**. The **map** prefix is currently defined as http://cs.vu.nl/eculture/map/. If this property is associated to a class, an XML element with the defined name is translated into an instance of this class. If it is associated to a property, it affects XML attribute or atomic element translation in two ways:

- It uses the RDF property name rather than the XML name for the property
- The rdfs:range of the property affects the value translation:
    - If it is rdfs:XMLLiteral, the sub-element is translated to an RDF XMLLiteral.
    - If it is an XSD datatype, the sub-element is translated into a typed RDF literal
    - It it is a proper class and the value is a valid URI, the URI is used as value without translation into a literal.

Below is an example that maps XML elements record into vra:Work instances and maps the XML attribute title into the vra:title property. Note that it is not required (and not desirable) to add the map:xmlname properties to the actual schema files. Instead, put them in a separate file and load both into the conversion engine.

```
@prefix  vra: <http://www.vraweb.org/vracore/vracore3#> .
@prefix  map: <http://cs.vu.nl/eculture/map/> .

# Map element-names to rdf:type
vra:Work map:xmlname "record" .

# Map xml attribute and sub-element names to properties
vra:title map:xmlname "title" .
```

2.3 Default XML name mapping

The initial XML to RDF mapper uses the XML attribute and tag-names for creating RDF properties. It provides two optional processing steps that make identifiers fit better with the RDF practice.

1. It can add a *prefix* to each XML name to create a fully qualified URI. E.g.,

```
?- rdf_current_ns(ahm, Prefix),
   load_xml_as_rdf('data.xml',
                   [ prefix(Prefix)
                   ]).
```

2. It ‘restyles' XML identifiers. Notably identifiers that contain a dot (.) are hard to process using Turtle. The library identifies alphanumerical substrings of the XML name and constructs new identifiers from these parts. By default, predicates start with a lowercase letter and each new part starts with an uppercase letter, as in `oneTwo`. Types (classes) start with an uppercase letter, as in `OneTwo`. This behaviour can be controlled with the options `predicate_style` and `class_style` of **load_xml_as_rdf**/2.

2.4 Subsequent meta-data mapping

Further mapping of meta-data consists of the following steps:

1. Fix the node-structure.
2. Re-establish internal links.
3. Re-establish external links.
4. Create a mapping schema that link the classes and predicates of the source to the target schema (e.g., Dublin Core).
5. Assign URIs to blank nodes where applicable.

Fix the node-structure

Source-data generally uses a record structure. Sometimes, each record is a simple flat list of properties, while in other cases it has a deeply nested structure. We distinguish three types of properties:

1. Properties with a clear single literal value, such as a collection-identifier. Such properties are directly mapped to RDF literals.
2. Properties with instance-specific scope that may have annotations. Typical examples are the title (multiple, translations, who has given the work a title, etc.) or a dimension (unit, which dimension, etc.). In this case, we create a new RDF node for each instance.
3. Properties that link to external resources: persons (creator), material (linking to a controlled vocabulary), etc. In this case the mapper unites multiple values that have the same properties. E.g., we create a single creator node for all creators found in a collection that have the same name a date of birth.

Unfortunately, (2) and (3) may be combined in one node. In that case the attributes that describe the node must first be separated from the attributes that refine the relation to the parent node.

For cases (2) and (3) above, each metadata field has zero or more RDF nodes that act as value. The principal value is represented by rdf:value, while the others use the original property name. E.g., the AHM data contains

```
Record title Title .
Record title.type Type .
```

This is translated into

```
Record title [ a ahm:Title ;
        rdf:value "Some title" ;
        ahm:titleType Type ;
       ] .
```

If the work has multiple titles, each title is represented by a separate node.

Because this step may involve using ordering information of the initial XML data that is still present in the raw converted RDF graph, this step must be performed before the data is saved.

2.4.2 Re-establish internal links

This step is generally trivial. Some properties represent links to other works in the collection. The property value is typically a literal representing a unique identifier to the target object such as the collection identifier or a database key. This step replaces the predicate-value with an actual link to the target resource.

2.4.3 Re-establish external links

This step re-establishes links from external resources such as vocabularies which we know to be used during the annotation. In this step we only make mapping for which we are *absolutely* sure. I.e., if there is any ambiguity, which is not uncommon, we maintain the value as a blank node created in step (1).

2.4.4 Create a mapping schema

It is advised to maintain the original property- and type-names (classes) in the RDF because this

1. Allows to reason about possible subtle differences between the source-specific properties and properties that come from generic schemas such as Dublin Core. E.g., a creator as listed for a work in a museum for architecture is typically an architect and the work in the museum is some form of reproduction on the real physical object. If we had replaced the original creator property by `dcterms:creator`, this information is lost.

2. It makes it much easier to relate the RDF to the original collection data. One of the advantages of this is that it becomes easier to reuse the result of semantic enrichment in the original data-source.

This implies that the converted data is normally accompagnied by a schema that lists the properties and types in the data and relates them using rdfs:subPropertyOf or rdfs:subClassOf to one or more generic schemas (e.g., Dublic Core). ClioPatria provides a facility to compute a schema for a graph from the actual data. This schema can be used as a starting point. To get this schema, open the ClioPatria web-interface, Use **Places/Graphs** to locate the graph and choose the option *Compute a schema for this graph and **Show** the result as **Turtle***.

2.4.5 Assign URIs to blank nodes where applicable.

Any blank node we may wish to link to from the outside world needs to be given a real URI. The record-URIs are typically created from the collection-identifier. For other blank nodes, we look for distinguishing (short) literals.

2.5 Enriching the crude RDF

The obtained RDF is generally rather crude. Typical 'flaws' are:

- It contains literals where it should have references to other RDF instances.
- One probably wants proper resources for many of the blank nodes.
- Some blank nodes provide no semantic organization and must be removed.
- At other place, intermediate instances must be created (as blank nodes or named instances).
- In addition to the above, some literal fields need to be rewritten, sometimes to (multiple) new literals and sometimes to a named or bnode instance.

Our rewrite language is a production-rule system, where the syntax is modelled after CHR (a committed-choice language for constraint programming) and the triple notation is based on Turtle/SPARQL. There are 3 types of production rules:

- **Propagation rules** add triples
- **Simplication rules** delete triples and add new triples.
- **Simpagation rules** are in between. They match triples, delete triples and add triples,

The overall syntax for the three rule-types is (in the order above):

```
<name>? @@ <triple>* ==> <guard>? , <triple>*.
<name>? @@ <triple>* <=> <guard>? , <triple>*.
<name>? @@ <triple>* \ <triple>* <=> <guard>? , <triple>*.
```

Here, <guard> is an arbitrary Prolog goal. <triple> is a triple in a Turtle-like, but Prolog native, syntax:

```
{ <subject> , <predicate> , <object> }
```

Any of these fields may contain a variable, written as a Prolog variable: an uppercase letter followed by zero or more letters, digits or the underscore. E.g., Hello, Hello_world, A9. Resources are either fully (single-)quoted Prolog atoms (E.g. 'http://example.com/me', or terms of the form <prefix> : <local>, where <prefix> is a defined prefix (see rdf_register_ns/2) and <local> is a possible quoted Prolog atom. E.g., vra:title or ulan:'Person' (note the quotes to avoid interpretation as a variable). Literals can use a more elaborate syntax:

```
<string> ^^ <type>
<string> @ <lang>
<string>
literal(Atom)
```

Here, <string> is a double-quoted Prolog string and <type> is a resource. The form literal(Atom) can be used to match the text of an otherwise unqualified literal with a variable. I.e.,

```
{ S, vra:title, literal(A) }
```

has the same meaning as the SPARQL expression ?S vra:title ?A FILTER isLiteral(?A),

Triples in the *condition* side can be postfixed using '?', in which case they are optional matches. If the triple cannot be matched, triples on the production-side that use the variable are ignored.

Triples in the *condition* can also be enclosed in a Prolog list ([...]), In this case, the triples are requested to be in the **order** specified. Ordering is not an official part of the RDF specs, but the SWI-Prolog RDF store maintains the order of triples in generated in the XML conversion process. An ordered set can match multiple times on a given subject, where it AB can match both AAABBB and ABABAB. Both forms appear in real-world XML data.

Finally, on the *production* side, the *object* can take this form:

```
bnode([ {<predicate> = <object>}
    ],
    [ {<option>}
    ])
```

This means, ‘for the object, create a bnode from the given <predicate> = <object> pairs'. The <option>s guide the process. At this moment, there is only one option with two values:

```
share_if(equal)
share_if(equal([<predicate>*]))
```

Without any option, each execution of the rule creates a new bnode. With the share_if option equal, it uses the same bnode-id for all productions that produce the same predicate-object list (in canonical order, after removing duplicates). Using the last form, it considers two blank nodes equal if they have the same triples on the given predicates. All other predicates are simply added to the blank-node.

2.5.1 Renaming resoures (or naming blank-nodes)

The construct {X} can be used on the condition and action side of a rule. If used, there must be exactly one such construct, one for the resource to be deleted and one for the resource to be added. All resources for which the condition matches are renamed. Below is an example rule. The first triple extracts the identifier. This triple must remain in the database. The \ {A} binds the (blank node) identifier to be renamed. The two Prolog guards verify that the resource is a blank node and generate an identifier (URI). The *action* (*{S}*) gives the rule engine the URI that must be given to the matched {A}.

```
work_uris @@
{ A, vra:'idNumber.currentRepository', ID } \ {A} <=>
  rdf_is_bnode(A),
  literal_to_id(ID, ahm, S),
  {S}.
```

5.2 Putting triples in another graph

Triples created by the *action* side of a rule are added to the graph that is being rewritten. It is also possible to add them to another graph using the syntax below:

```
  { S,P,O } >> Graph
```

E.g., if we want to store the information about person resources that we create in a graph named persons, we can so so using a rule like this:

```
person @@
{S, creator, Name},
{S, 'creator.date_of_birth', Born} ?,
{S, 'creator.date_of_death', Died} ?,
{S, 'creator.role', Role} ?
    <=>
    Name \== "onbekend",
    name_to_id(Name, ahm, Creator),
    { S, vra:creator, Creator },
    { Creator, rdf:type, ulan:'Person' } >> persons,
    { Creator, vp:labelPreferred, Name } >> persons,
    { Creator, ulan:birthDate, Born }  >> persons,
    { Creator, ulan:deathDate, Died }  >> persons,
    { Creator, ulan:role, Role }          >> persons.
```

5.3 Utility predicates

The rewriting process is often guided by a *guard* which is, as already mentioned, an arbitrary Prolog goal. Because translation of repositories shares a lot of common tasks, we plan to develop a library for these. This section documents the available predicates.

`[det]`**`literal_to_id`**`(`*`+LiteralOrList, +NS, -ID`*`)`

Generate an identifier from a literal by mapping all characters that are not allowed in a (Turtle) identifier to _. *LiteralOrList* can be a list. In this case we generate an id for each element in *LiteralOrList* and append these. A typical usage scenario is to add a type:

```
literal_to_id(['book-', Literal], NS, ID)
```

Another is to add the label of the parent:

literal_to_id([ParentLit, '-', Literal], NS, ID)

To be done

- Verify that the generated URI is unique!
- Remove diacritics for non-iso-latin-1 text

2.6 Putting it all together (examples)

Below we give some rules that we wrote to convert real data.

2.6.1 Deleting a triple

Sometimes XML contains data that simply means `nothing'. We want to delete this data:

```
{_, creator, "onbekend" } <=>
  true.
```

Now, in the data from which this was extracted, this is a bit too crude because some records keep data about the creator even though his/her name is not known. Therefore, we preceed the rule with the rule of the next section. Note that the order of rules matter: a rule is executed before the next one. In this particular case we could have removed the {S, creator, "onbekend"} triple from the example below to make it match after the rule above is executed.

2.6.2 Preserving info about unknown creators

The example below deals with entries in the database where the ‘creator' is unknown (Dutch: *onbekend*), but some properties are known about him or her. The remainder of the condition matches possible information about this creator using an ‘optional' match. The *guard* verifies there is at least some information about our unknown creator. The *action* part of the rule associates a new blank node as a creator.

```
creator_onbekend @@
{S, creator, "onbekend"},
{S, 'creator.date_of_birth', Born} ?,
{S, 'creator.date_of_death', Died} ?,
{S, 'creator.role', Role} ?
    <=>
    at_least_one_given([Born, Died, Role]),
    { S, vra:creator,
     bnode([ ulan:birthDate = Born,
         ulan:deathDate = Died,
         ulan:role = Role
        ])
    }.

at_least_one_given(Values) :-
    member(V, Values),
    ground(V), !.
```

2.6.3 Negation

Negation is only provided as Prolog negation--by-failure in the guard. This implies that we cannot use the {...} triple notation to test on the absence of a triple, but instead we need to use the SWI-Prolog RDF-DB primitive rdf/3. For example, to delete all person records that have no name, we can use the rule below. The first triple verifies the record-type. The second matches all triples on that record and the guard verifies that the subject has no triples for the property ahm:name.

```
delete_no_name @@
{ S, rdf:type, ahm:'Person' },
{ S, _, _ }
    <=>
    \+ rdf(S, ahm:name, _).
```

2.7 Running the toolkit

Installing the xmlrdf package, creates the directory cpack/xmlrdf/examples/AHM, containing the code we used to convert the Amsterdam Museum data. It contains three examples, one for converting the *people* database, one for the *thesaurus* and a one for the meta-data about the collection. Each conversion consists of a *run* file (e.g. **run-people.pl**) and a corresponding *rule* file (e.g. **rewrite-people.pl**) The *run* file loads relevant background data and defines **run/0** to call the initial converter. The options specify that the input is XML without namespaces (dialect xml rather than xmlns) and that the file contains XML elements named record as the desired unit of data for conversion.

```
run(File) :-
    load_xml_as_rdf(File,
            [ dialect(xml),
             unit(record)
            ]).
```

The result can be viewed by directing your browser to http://localhost:3020/, where 320 is the default port used by ClioPatria.

The file *rule* file scripts the rewrite phase. It sets up namespace prefixes, calls to the rewrite predicates with the proper arguments and finally provides the rules. Here are the toplevel predicates:

`rewrite`

Apply all rules on the graph `data`

`rewrite``(`*`+Rule`*`)`

Apply the given rule on the graph `data`

`rewrite``(`*`:To, +From`*`)`

Invoke the term-rewriting system

`rewrite``(`*`+Graph, +Rule`*`)`

Apply the given rule on the given graph.

`list_rules`

List the available rules to the console.

Below is an example run, showing all available rules and running a single rule. The example demonstrates that rules are applied until a fixed-point is reached (i.e., the RDF database does not change by applying the rules).

```
?- [rewrite].
true.

?- list_rules.
Defined RDF mapping rules:

    title_translations
    dimension
    work_uris
    creator_sequence
    creator_onbekend
    delete_unknown_creator
    delete_empty_literal
    creator
    material_aat
    related_object

true.

?- rewrite(delete_empty_literal).
% Applying ... delete_empty_literal (1)
% 0.100 seconds; 23,456 changes; 2,008,860 --> 1,985,404 triples
% Step 1: generation 2,020,746 --> 2,044,202
% Applying ... delete_empty_literal (1)
% 0.000 seconds; no change
% Step 2: generation 2,044,202 --> 2,044,202
true.
```

2.8 Trips and Tricks

Here follow a few hints for working with the XMLRDF tool which arouse from several proofing exercises executed at HUB which included the writing of conversion scripts.

Preparing your data

Preparing your source data with either, for example, XSLT before loading the data into the XMLRDF tool or after loading the data into the XMLRDF tool before manipulating the actual structure can make further processing of the data structure much easier. Such preparation could be deleting superfluous levels in a complex XML hierarchy. Or you could rename XML tags so that it is easier to match specific parts within the generic RDF data: In the case of METS files, for example, there is a section which describes the logical and one which describes the physical structure of a complex object. Both section have ID tags. If you rename those tags to, for example, “ID_Logical” and “ID_Physical” it is easier to match those triples with simpler rules. By preparing your data in such way you avoid too complex rules or additional Prolog clauses, i.e. you do not have to go outside the XMLRDF rule system.

Write and test simple rules one after the other

Write and test one rule after the other. Execute a rule you wrote and test if it works and also check how the structure of the source data changes. If a rule is executed it immediately effects the cached source data and therefore affects the following rules.

It is easier to write small rules which only perform a single, limited restructuring task. It is easier to keep track of what is happening and easier to change rules later, i.e. add and delete rules.

Common “orthographical” and other errors to look out for

Because the XMLRDF tool does not have a syntax checker here are some common errors to look out for if an error is reported after compiling rewrite.pl or a rule seems not to match the triples it should match.

Use simple quotation marks with names beginning with a capital letter:

```
{ BN1, rdf:type, sbb:'StructMap' }
```

Use double quotation marks with Literals:

```
{ BN1, sbb:'TYPE', "LOGICAL" }
```

Be careful about captial and non-capital letters. Variables start with a capital letter:

```
{ BN, sbb:'TYPE', Type } \
{ BN, rdf:type, sbb:'StructMap' }
<=>
  member(Type, ["LOGICAL", "PHYSICAL"]),
  literal_to_id(['structMap', Type], sbb, StructMap),
  { BN, rdf:type, StructMap }.
```

Remember the commata between the rules and the full stop after the last rule in your clause:

```
{ URIaggregation, ens:aggregatedCHO, URIphysical },
{ URIproxy, dct:title, LABEL }.
```

The question mark stands before the commata (or full stop at the end of the clause):

```
{ BN_4, sbb:titleInfo, BN_5 } ?,
```

Build the triangle

Proposal of rules to create the basic triangle (or just the Aggregation and CHO leaving the Proxy out) and to create the URIs.

```
% match the nescessary triples and values here

==>

  % create URI for the proxy
  literal_to_id(['proxy-', RecordID, '-', ID], ns, URIproxy),
  % create URI for the aggregation
  literal_to_id(['aggregation-', RecordID, '-', ID], sbb,
URIaggregation),
  % create URI for the object of interest
  literal_to_id(['cho-', RecordID, '-', ID], ns, URIcho),

  % add ore:Proxy
  { URIproxy, rdf:type, ore:'Proxy' },
  % add ore:Aggregation
  { URIaggregation, rdf:type, ore:'Aggregation' },
  % add edm:ProvidedCHO
  { URIcho, rdf:type, edm:'ProvidedCHO' },

  % link ore:Proxy and edm:ProvidedCHO by ore:proxyFor
  { URIproxy, ore:proxyFor, URIcho },
  % link ore:Proxy and ore:Aggregation by ore:proxyIn
  { URIproxy, ore:proxyIn, URIaggregation },
  % link ore:Aggregation and edm:ProvidedCHO by ore:aggregates
  { URIaggregation, ore:aggregatedCHO, URIcho }.
```

3 AMALGAME documentation

Home page: http://semanticweb.cs.vu.nl/AMALGAME/

Part of this documentation has been published as (van Ossenbruggen, 2011) and in Europeana Milestone 1.2.2.

AMALGAME is implemented as a packge for ClioPatria. The home page of this pacakge is http://cliopatria.swi-prolog.org/packs/AMALGAME. The name of the package is **AMALGAME** and the package can thus be installed from ClioPatria using this command:

?- **cpack_install(AMALGAME).**

An implementation of AMALGAME is running on and can interact with the Semantic Layer at http://semanticweb.cs.vu.nl/europeana/.

3.1 Introduction

AMALGAME (AMsterdam ALignment GenerAtion MEtatool) is a tool for finding, evaluating and managing vocabulary alignments. We explicitly no dot aim to produce 'yet another alignment method' but rather seek to combine existing matching techniques and methods such as those developed within the context of the Ontology Alignment Evaluation Initiative (OAEI[1]), in which different alignment methods can be combined using a workflow setup (van Ossenbruggen, 2011).

AMALGAME features:

- An interactive *workflow composition functionality*. Using this setup, the user can iteratively select various actions on vocabularies or intermediate mapping results. Filters can be applied to select subsets of concepts or of mapping results. By concatenating these actions, an alignment workflow emerges.
- A *statistics function*, where statistics for intermediate and end-result alignment sets are be shown.
- An *evaluation view,* where subsets of alignments can be evaluated manually.

3.2 Requirements analysis

The requirements for the AMALGAME alignment platform have been based on previous research (Tordai et al 2009, 2010a, 2010b) as well as feedback from alignment efforts in the EuropeanaConnect, PrestoPrime and MultimediaN E-culture projects (van Ossenbruggen, 2011).

First, domain experts find it hard to choose which tool to select for their alignment task. From the alignment research literature, it is clear how each tool performs on the data used in the evaluation experiments. However, it is hard for experts to predict which tool would perform well on their own data set.

Second, experts perceive the current tools to not support the large and shallow vocabularies that are typical for their domain. Alignment runs take either a very long time or do not finish at all.

[1] http://oaei.ontologymatching.org/

Third, when an alignment run finishes, it typically produces a result set with a large number (e.g. over 100k) of correspondences, but provide little support to assess the quality of these results. Furthermore, the quality of the correspondences might not be homogeneously distributed across the alignment result set. Different subsets of alignments might have different features that determine the quality of the end result. Transparent and interactive assessment is crucial to be able to decide whether the result is of sufficient quality.

Fourth, when the results are not sufficient it is unclear how the tool should be (re-)configured to improve the results. Experts need to be able to understand why the tool found erroneous correspondences and how to get rid of them in a next step. When the tool failed to find correct correspondences, the experts need to know how to find those in a next step. This requires insight in how the alignment algorithms work, and how to configure them to adjust them to the specific needs of vocabularies at hand.

Based on this feedback, we conclude that in our domain, the ability to quickly run simple matching algorithms in an interactive environment that supports analysis of large sets of correspondences has the potential to directly solve the problems mentioned above. The current AMALGAME version purely relies on (human) expertise for configuring the right sequence of operations. Future versions may incorporate NLP or machine learning techniques to help the human expert it selecting representative subsets for evaluation and/or suggesting (parts of) a plan. An interactive alignment environment needs to fulfil the following requirements that would distinguish it from most current alignment toolkits:

The resulting key requirements for AMALGAME are thus:

- *Speed*. To allow interactive scenarios, it is important that the matchers are sufficiently fast to be run in an interactive setting. To align large vocabularies in an interactive environment, computationally cheap matching strategies will work better than the more expensive ones used by most alignment tools today.
- *Transparancy.* In an interactive setting, it is also important that users understand each intermediate result to be able to judge how to improve it in a next step. They would favour simple matching algorithms in which the pros and cons are easy to understand over complex ones for which domain experts cannot explain the results. Domain experts often have a deep understanding about the characteristics of their vocabularies. If a tool also allows them to understand the matching techniques used, they should have sufficient knowledge to make informed choices about the design of an alignment strategy that is targeted to the specific needs of their own data.
- *Interactive*: users should be able to run alignments interactively (via a web-based interface).
- *Configurable components and overall workflow*. Assuming that the user indeed has sufficient knowledge about the data and the available matching algorithms to design a targeted alignment strategy, we need a method in which the user can tune the parameters of each matcher to the specific needs of her data set. In addition, the user also needs to be able to configure which matchers to run for which part of the data and in which order.
- *Vocabulary and result analysis*. Assuming that the above requirements are met, users can quickly configure and run different matchers on even large data sets. The most expensive step in such an interactive environment would then no longer be finding the

correspondences, but the analysis of the large result sets (and optionally an re-analysis of the vocabularies) to decide on which step to take next. Tool support for the analysis of the large sets of correspondences that are typical for this domain is thus also crucial.

- *Provenance*. An additional requirement follows from the potential for the use of the resulting alignments in other application scenarios. Within the context of the Semantic Web and Linked Open Data, more and more alignments are being published on the Web and reused in a widely different set of contexts. For many of these alignments, it remains unclear how they have been generated, and how the same or a similar data set could be reproduced. For example, as new versions of the underlying vocabularies become available, one might want to update the associated alignments by rerunning the same alignment technique on the new versions. When alignments are the result of a scientific experiment described in a research paper, it also desirable to be able to replicate the experiment and have the results confirmed by others. However, most alignments currently published have insufficient metadata to allow alignments to be reproduced. Highly interactive alignment strategies as proposed in this paper have the potential to make this situation even worse. To address this, we require that an interactive alignment tool records sufficient information about each individual step, and the order in which the steps are executed, that the result set can be fully reproduced later.

3.3 The AMALGAME approach to vocabulary alignment

To address the requirements above, we developed an alignment approach that improves the speed and transparency of the alignment process by drastically reducing the complexity of the technology, allowing the user how to combine a limited number of basic building blocks into a alignment workflow targeted to the data set at hand. Each building block should be sufficiently simple to produce an understandable result. Which blocks to use and in what order or combination is fully controlled by the user. Furthermore, produced alignments – both intermediate and end results – can be easily and quickly evaluated to give insight in their quality. We have build a prototype alignment service that has been designed with this approach in mind, and used the prototype to create alignments in three different use cases, that will be discussed in the next section. Here we sketch an high level overview of the AMALGAME alignment methodology and will flesh out some interesting details in the context of the use case descriptions.

3.3.1 Analyze vocabularies

An assumption of the interactive approach is that the user has knowledge of the vocabularies being aligned. For SKOS-like vocabularies we identify two types of characteristics. First, the user needs to know the number concepts that the source and target vocabularies contain. For example, if vocabulary A is ten times the size of B, it is unlikely to find equivalence relations for more that 10\% of A's concepts, and looping over all concepts in B to find correspondences in A will be more efficient than vice versa. Large vocabularies are often heterogeneous, which makes it important that the user knows the types of concepts the vocabularies contain. By matching only similar types of source and target concepts, the workload is reduced and the precision can be increased. There is, for example, no need to align concepts representing persons with those representing locations. Second, the user has to identify the concepts' properties that can be used in the matching process. In SKOS-like vocabularies preferred and alternative labels are likely candidates for simple string matching techniques. Using alternative labels typically improves recall over using only preferred labels, but may also reduce precision. A vocabulary owner should be able to make her own trade off what to use in which case. Knowing which correspondences result from preferred label matches only and which not may also help in designing a more targeted evaluation strategy (e.g. by deciding to evaluate a relative smaller sample of preferred label matches) or by deciding what next steps to take (e.g. that the preferred label matches are of sufficient precision but that the alternative label matches need additional filtering to remove false positives). In addition, there may be labels in different languages. Typically, the user wants to avoid matching of labels that are in different languages to prevent false matches, but sometimes multilinguality can also be used as an advantage, for example when syntactic label match found in multiple languages can be interpreted as extra evidence for a given correspondence. Besides the labels, the vocabularies may provide other properties that can be matched. Typically, longer textual descriptions found in SKOS definitions or scope notes may provide another source of textual information. Hierarchical or associative relations provide structural information. These properties can be used to create new sets of alignments (boosting recall) but also to improve already existing alignments, for (boosting precision). For example, in geographical thesauri the hierarchical containment is often sufficient to distinguish between ambiguous label matches. Properties such as SKOS editorial notes are typically not useful for matching, but there are exceptions in individual cases. For example, we have came across several cases where editorial notes associated with many concepts in one vocabulary contained the unique identifier of the concept of another vocabulary from which the concept had once been copied. Such information obviously simplifies alignment drastically, but requires that the user knows these notes are present and that this knowledge can subsequently be used as input in the alignment workflow.

3.3.2 Generate matches

Our approach is to have the user interactively construct an alignment workflow. The individual building blocks of this workflow are elementary matching techniques. Combined with the knowledge about the vocabularies, acquired in the analysis phase, using simple processes should result in a predictable outcome. Which technique is most suited for the first step depends on the source and target vocabularies, and might vary for different types of concepts or properties within these vocabularies. Our method is independent of the specific matching technique used, but we assume each technique takes a specification for the source and target concepts as input and produces a set of correspondences. In addition, each technique can have parameters to tune it for the data at hand. We distinguish matching techniques that use textual properties from techniques that use structural information. The literature provides a wide range of similarity metrics to match textual properties. Our tool provides a number of these metrics out-of-the-box including exact label matching, prefix matching and jaccard matching.

Structural matching uses the position of a concept in the graph or tree structure of the vocabulary. Typical structural information in SKOS like vocabularies is provided by thehierarchical and associative relations, where the number of steps that are considered in the matching are typical parameters. Again, our tools provides reasonable defaults but leaves tuning to the user. The configuration parameters for the matching techniques vary per algorithm and while our tool provides reasonable defaults for each, we intentionally put the full burden of selecting the right parameters on the user.

Next to matching techniques, we also provide building blocks that represent filtering actions. There are two types of filters implemented. The first type is used to select a subset of concepts from a vocabulary based on some criterion.This includes a SKOS subtree, or the set of concepts that have a certain value for an object property. Using this filtering technique, users van for example select only geographic concepts from a general thesaurus, or focus on aligning a sub-domain.

The second filtering action takes as input an intermediate mapping. Here mappings can be selected or discarded based on a number of criteria, including whether or not the matches are ambiguous (one concept has a correspondence to multiple concepts). Likewise, matches with a confidence value below some threshold can be filtered out.

3.3.3 Analyze matches and iterate

Alignment in our approach is inherently an iterative process, the user needs to determine – after each matching technique that has been applied -- what to do next. This decision depends on the correspondences that are generated. AMALGAME provides support to analyse large sets of correspondences. When the user is confident that a set of matches are correct he can limit investigation to a small sample. The user can also choose a strategic sample based on the properties of the correspondences, for this we will give examples of this in the use cases below. Independent of the analysis technique used, we identify four typical scenarios, depending on the outcome of the analysis. The *first scenario* is that a user decides the results are no good at all, in which case all results are simply discarded after analysis. Assuming the technique used is sufficiently simple, the user will understand from the analysis what caused the failure and will be able to try another matching run, using another technique or a better configuration of the technique used in the previous run. The *second scenario* is that the results are good, but that recall is low. To improve recall, the user can proceed by matching only the concepts that have not yet been aligned. Note that this typically is a significantly smaller set, so the user may decide to deploy computationally more expensive matching techniques to improve recall in subsequent runs. The *third scenario* is that the results are good, but that precision is low. To improve precision, users need to find techniques that allow them to distinguish true from false correspondences. Such techniques include selections based on properties of the current result set. For example, users may decide to only use one to one and to discard other correspondences, or, to select the preferred label match for those concepts that are ambiguously mapped due to an alternative mapping. In other cases, the current result set may provide insufficient information to make such a selection, in which case additional matching techniques can be run to find extra evidence to discriminate true from false correspondences. Again, more expensive matchers can be used to boost precision for smaller subsets. The *fourth scenario* is that a user decides that the results are of sufficient quality, after which she exports them to the desired format and we consider the alignment task to be successfully finished. Finally, the *last scenario* is that the user finds the results of insufficient quality, but is out of options and does not know how they can be further improved, in which case we consider the alignment task to be failed. In practice, we found the first scenario useful to quickly try some alternative matchers, and to compare, analyse and discard the results, just to develop some intuition before the real alignment task starts. Many alignment tasks, including the first two use cases discussed below, are based on an iteration of the second and third scenario. Ideally, with each iteration the set of concepts that have to still be aligned (to improve recall) and the set of correspondences that still have to be filtered (to improve precision) decreases, or, if not, the user gains some knowledge to achieve this in the next step.

3.3.4 Evaluation Tool Features

An important part of AMALGAME is the alignment evaluation tool. Here the user is presented with matches from a given mapping subset. By evaluating a set of mappings the user can quickly get an overview of the quality of the produced mappings.

For each subset, the user is presented with samples of matches that are to be evaluated; these samples are produced with different orderings (random, alphabetical, ordered by confidence score etc.). Samples can be generated for the whole vocabulary, for specific subsets (belonging to a subtree, with confidence value greater than a threshold etc.). The size can be dependent on the required representativeness of the sample. Stratified sampling using different criteria such as the ones described in [Tordai et al. 2009] will be supported.

To be able to quickly evaluate large numbers of matches, an efficient user interface is required. In this UI, the prefLabel, altLabels, scopenotes, definition is displayed, as well as the context of both the source and target concepts (broader and narrower terms, siblings). Any found mappings between the context concepts are also displayed to help the user to appraise the match. Any annotated objects can also be displayed for this purpose.

If there are other matches found for the source and or target concepts, these ambiguous matches will also be shown in the interface, as this will improve the efficiency and quality of the evaluation process.

The future version of the evaluation tool will also feature meta-evaluator options (ie. the ability to verify or annotate other users' evaluations).

3.4 Features and Infrastructure of the implementation

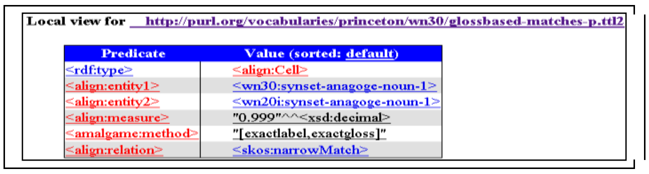
We here list the implemented features and infrastructure of AMALGAME.

3.4.1 Alignment formats

To represent mappings, we use the EDOAL (Expressive and Declarative Ontology Alignment Language) vocabulary.[2] This format is used for the alignment tasks within the OAEI (Ontology Alignment Evaluation Initiative) and serves as a de facto standard for representing fine-grained mappings in the Semantic Web community.

In the EDOAL format, an alignment is a set of mapping cells, each cell being a correspondence (mapping) between two entities. The alignment vocabulary extends this scheme by allowing cells to contain compound entity descriptions. Each entity can be typed according to one of the following category: Class, Instance, Relation, Property. A relation corresponds to an object property in OWL, a property to a datatype property. Each entity can then be restricted, and transformation can be specified on property values.

In AMALGAME, we store mapping cells using the RDF representation of EDOAL, which allows us to make use of the existing ClioPatria functionalities. An example of an EDOAL mapping cell in RDF Turtle syntax is given below. This match is the result of the overlap match of two separate methods that align the Art and Architecture Thesaurus (AAT) and WordNet version 2.0.



Local view for http://purl.org/vocabularies/princeton/wn30/glossbased-matches-p.ttl2

Predicate	Value (sorted: default)
<rdf:type>	<align:Cell>
<align:entity1>	<wn30:synset-anagoge-noun-1>
<align:entity2>	<wn20i:synset-anagoge-noun-1>
<align:measure>	"0.999"^^<xsd:decimal>
<amalgame:method>	"[exactlabel,exactgloss]"
<align:relation>	<skos:narrowMatch>

Figure 2: Example match between a Wordnet 3.0 and a Wordnet 2.0 concept in RDF EDOAL format

[2] http://alignapi.gforge.inria.fr/edoal.html

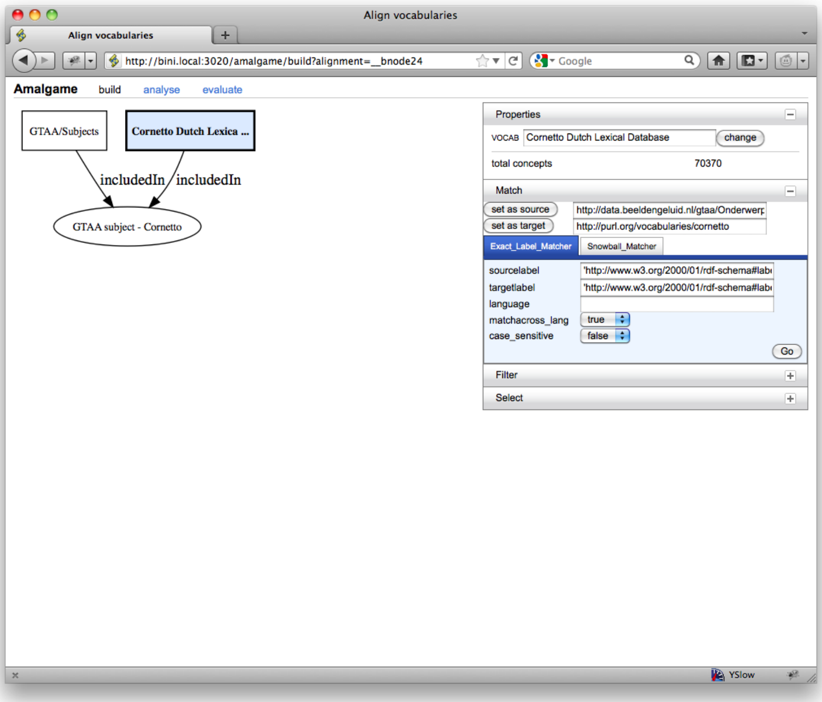
When using the OAEI data model to represent mapping cells, we explicitly re-use the SKOS relations exactMatch, closeMatch, broadMatch, narrowMatch and relatedMatch as relation types. This illustrates the complementarity between the two models: SKOS does not support the annotations of mappings as we need it; on the other hand, OAEI does not make any commitment with regards to the semantic type of relations that mappings assert. SKOS mapping properties support the appropriate types to fill this gap. [cf. van der Meij et al., 2010]

AMALGAME features alignment format rewriting library that can convert SKOS alignments to EDOAL format and 'concretify' EDOAL mappings to abovementioned SKOS formats. By being able to accepting multiple alignment formats, AMALGAME's functionalities (such as the evaluation tool) can be used for other, existing mappings.

3.4.2 Workflow builder

The AMALGAME user can at each point in the alignment process select different actions. These actions take one or more vocabularies and/or one an intermediate mappings as input and in turn produce either new mappings or filtered input sets. These different actions in the workflow are registered in an alignment provenance graph. This provenance graph is represented using the Open Provenance Model Vocabulary (OPMV) (http://code.google.com/p/opmv/).

An important feature of AMALGAME is that it is able to execute these provenance models. As such, they function both as a workflow specification and as a documentation. Using the cliopatria visualization capabilities, the provenance graphs are displayed to the user. Figure 3 and 4 show a screenshot of the AMALGAME alignment builder. In Figure 4 a finalized alignment provenance graph is displayed.



http://bini.local:3020/amalgame/build?alignment=__bnode24

VOCAB Cornetto Dutch Lexical Database

total concepts 70370

set as source http://data.beeldengeluid.nl/gtaa/Onderwerp

set as target http://purl.org/vocabularies/cornetto

sourcelabel 'http://www.w3.org/2000/01/rdf-schema#labe

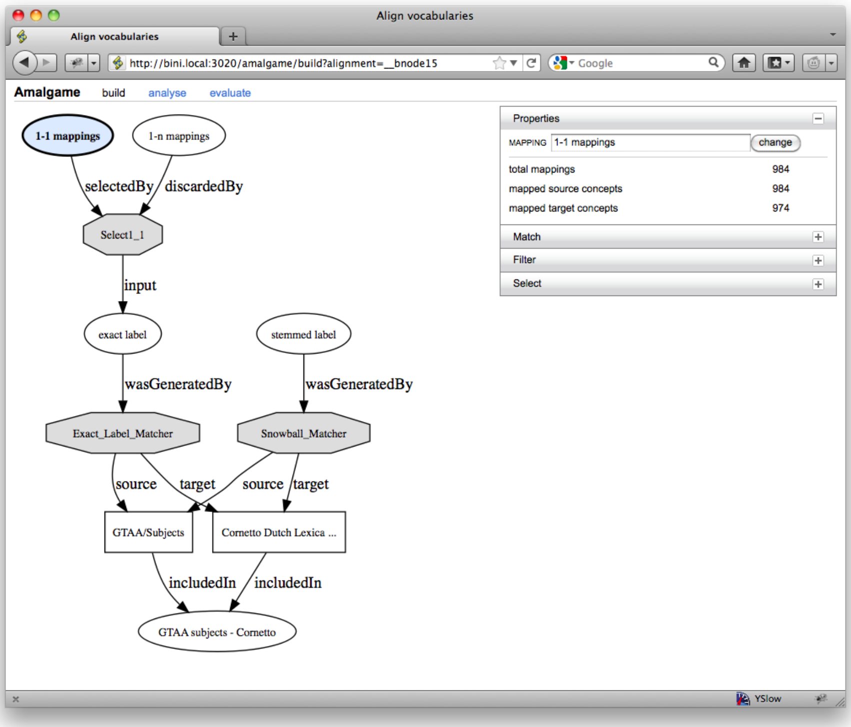
targetlabel 'http://www.w3.org/2000/01/rdf-schema#labe

language

matchacross_lang true

case_sensitive false

Figure 3: Screenshot of the AMALGAME alignment builder: selecting and configuring an action



Properties

MAPPING 1-1 mappings change

total mappings	984
mapped source concepts	984
mapped target concepts	974

Match

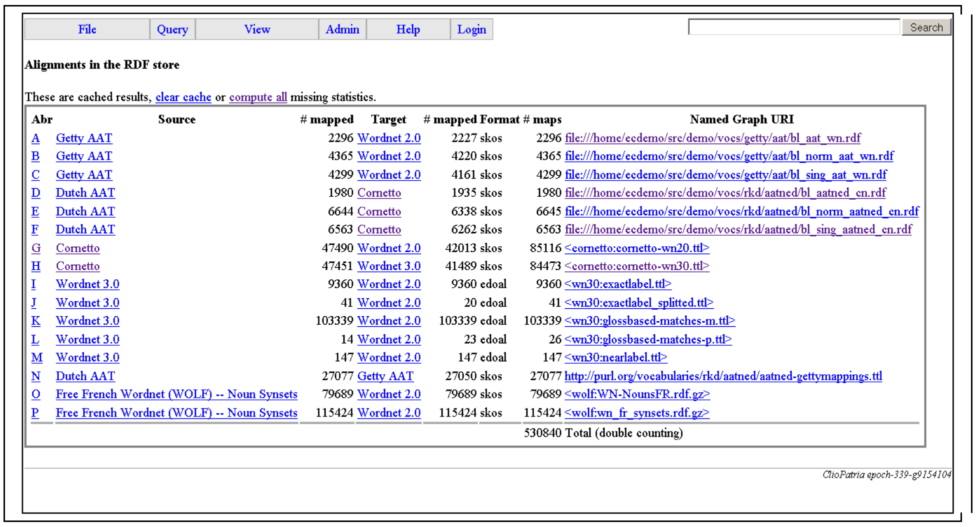
Filter

Select

Figure 4: Screenshot of the AMALGAME alignment builder: resulting alignment provenance graph

3.4.3 Statistics Features

In the current implementation of AMALGAME different alignments between vocabularies are loaded in RDF format (either SKOS mappings or EDOAL) and a number of statistics can be viewed. These include the total number of source concepts matched, the total number of target concepts matched and the total number of mapping cells that make up that alignment. The user can also browse to the source vocabulary, the target vocabulary or the mapping graph to view the individual graph statistics.



Alignments in the RDF store

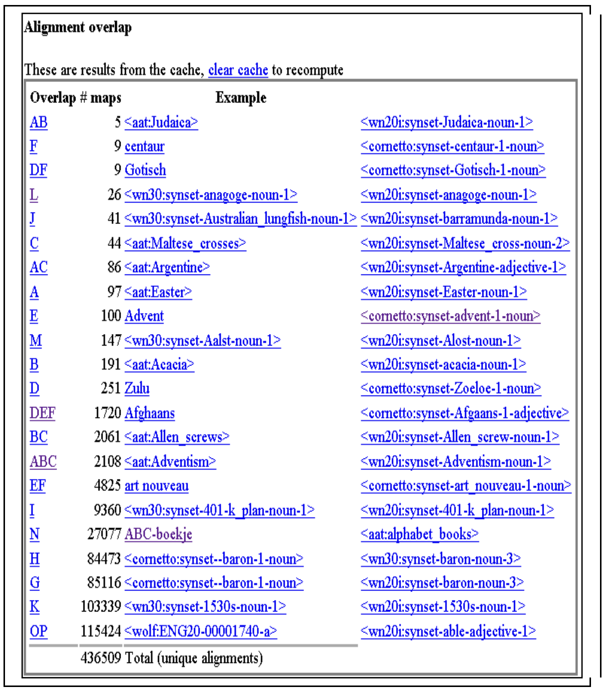
These are cached results, clear cache or compute all missing statistics.

Abr	Source	# mapped	Target	# mapped	Format	# maps	Named Graph URI
A	Getty AAT	2296	Wordnet 2.0	2227	skos	2296	file:///home/ecdemo/src/demo/vocs/getty/aat/bl_aat_wn.rdf
B	Getty AAT	4365	Wordnet 2.0	4220	skos	4365	file:///home/ecdemo/src/demo/vocs/getty/aat/bl_norm_aat_wn.rdf
C	Getty AAT	4299	Wordnet 2.0	4161	skos	4299	file:///home/ecdemo/src/demo/vocs/getty/aat/bl_sing_aat_wn.rdf
D	Dutch AAT	1980	Cornetto	1935	skos	1980	file:///home/ecdemo/src/demo/vocs/rkd/aatned/bl_aatned_cn.rdf
E	Dutch AAT	6644	Cornetto	6338	skos	6645	file:///home/ecdemo/src/demo/vocs/rkd/aatned/bl_norm_aatned_cn.rdf
F	Dutch AAT	6563	Cornetto	6262	skos	6563	file:///home/ecdemo/src/demo/vocs/rkd/aatned/bl_sing_aatned_cn.rdf
G	Cornetto	47490	Wordnet 2.0	42013	skos	85116	<cornetto:cornetto-wn20.ttl>
H	Cornetto	47451	Wordnet 3.0	41489	skos	84473	<cornetto:cornetto-wn30.ttl>
I	Wordnet 3.0	9360	Wordnet 2.0	9360	edoal	9360	<wn30:exactlabel.ttl>
J	Wordnet 3.0	41	Wordnet 2.0	20	edoal	41	<wn30:exactlabel_splitted.ttl>
K	Wordnet 3.0	103339	Wordnet 2.0	103339	edoal	103339	<wn30:glossbased-matches-m.ttl>
L	Wordnet 3.0	14	Wordnet 2.0	23	edoal	26	<wn30:glossbased-matches-p.ttl>
M	Wordnet 3.0	147	Wordnet 2.0	147	edoal	147	<wn30:nearlabel.ttl>
N	Dutch AAT	27077	Getty AAT	27050	skos	27077	http://purl.org/vocabularies/rkd/aatned/aatned-gettymappings.ttl
O	Free French Wordnet (WOLF) -- Noun Synsets	79689	Wordnet 2.0	79689	skos	79689	<wolf:WN-NounsFR.rdf.gz>
P	Free French Wordnet (WOLF) -- Noun Synsets	115424	Wordnet 2.0	115424	skos	115424	<wolf:wn_fr_synsets.rdf.gz>
						530840	Total (double counting)

ClioPatria epoch-339-g9154104

Figure 5: Screenshot of the Alignment viewing tab in AMALGAME/ClioPatria

The overlap of the different alignments can also be calculated. The overlap sets contain the mappings that occur in two or more sets. These are then stored as separate RDF graphs and displayed in the browser. Also displayed are the sets of mapping cells that occur in exactly one alignment. The total number of matches is displayed per overlap set as well as one example of a match in an overlap set [Figure 6].



Alignment overlap

These are results from the cache, clear cache to recompute

Overlap	# maps	Example	
AB	5	<aat:Judaica>	<wn20i:synset-Judaica-noun-1>
F	9	centaur	<cornetto:synset-centaur-1-noun>
DF	9	Gotisch	<cornetto:synset-Gotisch-1-noun>
L	26	<wn30:synset-anagoge-noun-1>	<wn20i:synset-anagoge-noun-1>
J	41	<wn30:synset-Australian_lungfish-noun-1>	<wn20i:synset-barramunda-noun-1>
C	44	<aat:Maltese_crosses>	<wn20i:synset-Maltese_cross-noun-2>
AC	86	<aat:Argentine>	<wn20i:synset-Argentine-adjective-1>
A	97	<aat:Easter>	<wn20i:synset-Easter-noun-1>
E	100	Advent	<cornetto:synset-advent-1-noun>
M	147	<wn30:synset-Aalst-noun-1>	<wn20i:synset-Alost-noun-1>
B	191	<aat:Acacia>	<wn20i:synset-acacia-noun-1>
D	251	Zulu	<cornetto:synset-Zoeloe-1-noun>
DEF	1720	Afghaans	<cornetto:synset-Afgaans-1-adjective>
BC	2061	<aat:Allen_screws>	<wn20i:synset-Allen_screw-noun-1>
ABC	2108	<aat:Adventism>	<wn20i:synset-Adventism-noun-1>
EF	4825	art nouveau	<cornetto:synset-art_nouveau-1-noun>
I	9360	<wn30:synset-401-k_plan-noun-1>	<wn20i:synset-401-k_plan-noun-1>
N	27077	ABC-boekje	<aat:alphabet_books>
H	84473	<cornetto:synset--baron-1-noun>	<wn30:synset-baron-noun-3>
G	85116	<cornetto:synset--baron-1-noun>	<wn20i:synset-baron-noun-3>
K	103339	<wn30:synset-1530s-noun-1>	<wn20i:synset-1530s-noun-1>
OP	115424	<wolf:ENG20-00001740-a>	<wn20i:synset-able-adjective-1>
	436509	Total (unique alignments)	

Figure 6: Screenshot of the alignment overlap browser

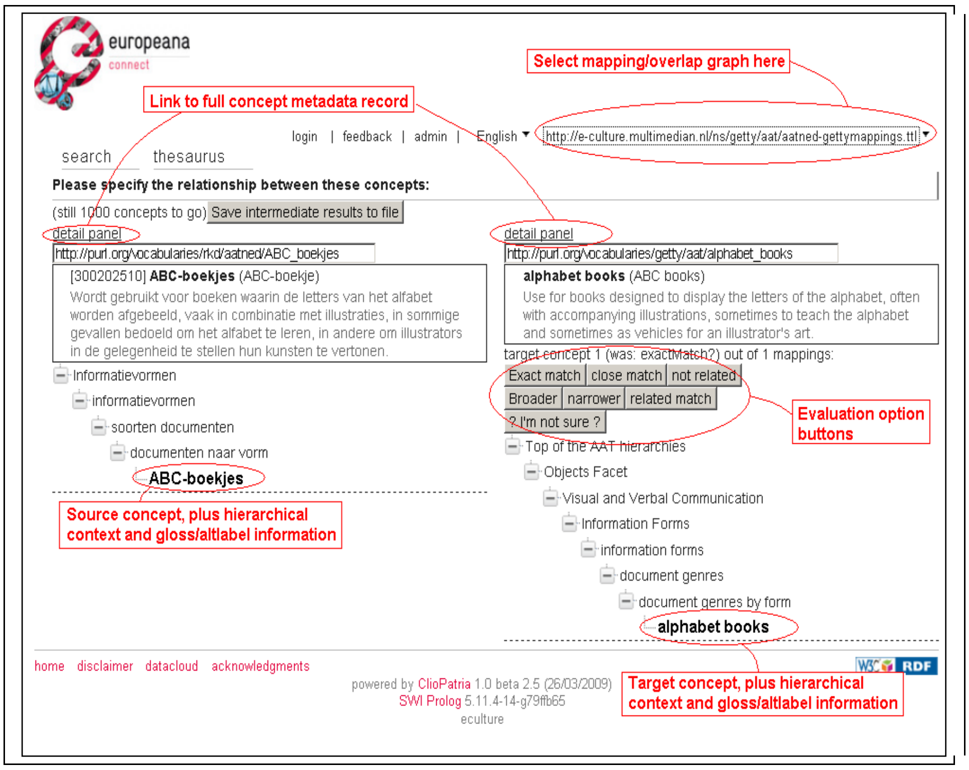
By clicking on an overlap set, users can browse to that graph, view its statistics (number of classes, predicates, subjects, blank nodes) and display or download the RDF graph.

3.4.4 Evaluation Features

The current implementation of the evaluation tool is displayed in Figure 7. The user can select one loaded alignment or overlap graph and is presented with individual matches. In the current version these mappings are presented alphabetically, more orderings corresponding to the requirements mentioned above will be implemented.

The hierarchical context (broader and narrower concepts in SKOS) is presented to the user, as well as gloss information and any alternative labels. Additionally, the user can choose to view the detail panel, where other properties and any annotated resources can be viewed.

The user can select one of seven evaluation options (exact match, close match, not related, broader, narrower, related match or "I'm not sure"). Through these buttons a suggested mapping can be acknowledged, further specified, discarded or ignored. The intermediate results can be stored as RDF graphs.


europeana
connect
Select mapping/overlap graph here
Link to full concept metadata record
login | feedback | admin | English
search
thesaurus
Save intermediate results to file
detail panel
detail panel
Exact match
close match
not related
Broader
narrower
related match
? I'm not sure ?
Evaluation option buttons
Source concept, plus hierarchical context and gloss/altlabel information
Target concept, plus hierarchical context and gloss/altlabel information
home
disclaimer
datacloud
acknowledgments
W3C RDF

http://e-culture.multimedian.nl/ns/getty/aat/aatned-gettymappings.ttl

Please specify the relationship between these concepts:

(still 1000 concepts to go)

http://purl.org/vocabularies/rkd/aatned/ABC_boekjes

[300202510] **ABC-boekjes** (ABC-boekje)
Wordt gebruikt voor boeken waarin de letters van het alfabet worden afgebeeld, vaak in combinatie met illustraties, in sommige gevallen bedoeld om het alfabet te leren, in andere om illustrators in de gelegenheid te stellen hun kunsten te vertonen.

- Informatievormen
  - informatievormen
    - soorten documenten
      - documenten naar vorm
        - **ABC-boekjes**

http://purl.org/vocabularies/getty/aat/alphabet_books

alphabet books (ABC books)
Use for books designed to display the letters of the alphabet, often with accompanying illustrations, sometimes to teach the alphabet and sometimes as vehicles for an illustrator's art.

target concept 1 (was: exactMatch?) out of 1 mappings:

- Top of the AAT hierarchies
  - Objects Facet
    - Visual and Verbal Communication
      - Information Forms
        - information forms
          - document genres
            - document genres by form
              - **alphabet books**

powered by ClioPatria 1.0 beta 2.5 (26/03/2009)
SWI Prolog 5.11.4-14-g79ffb65
eculture

Figure 7: Annotated screenshot of the current implementation of the alignment evaluation tool

References

- A. Tordai, J. van Ossenbruggen and G. Schreiber. Combining Vocabulary Alignment Techniques. In K-CAP '09: Proceedings of the 5th international conference on Knowledge capture, pages 25–32, 2009. ACM.
- A. Tordai, J. R. van Ossenbruggen, A. Ghazvinian, M. A. Musen, and N. F. Noy. Lost In Translation? Empirical Analysis Of Mapping Compositions For Large Ontologies. In Proceedings of International Workshop on Ontology Matching 2010 (5). CEUR-WS, November, 2010. [Tordai et al. 2010a]
- A. Tordai, J. R. van Ossenbruggen, G. Schreiber, and B. Wielinga. Aligning Large SKOS-Like Vocabularies. In Proceedings of European Semantic Web Conference 2010 (7), Lecture Notes in Computer Science, pages 198–212. Springer, May 2010. [Tordai et al. 2010b]
- Lourens van der Meij, Antoine Isaac, and Claus Zinn. A web-based repository service for vocabularies and alignments in the Cultural Heritage domain. In proceedings of the 7th Extended Semantic Web Conference, (ESWC 2010). Heraklion, Greece, 30 May–3 June 2010.
- Jacco van Ossenbruggen, Victor de Boer, Michiel Hildebrand and Antoine Isaac. Vocabulary alignment as an interactive and replicable workflow. PrestoPrime deliverable D.4.2.1. semanticweb.cs.vu.nl/lod/prestoprimeD421/paper.pdf February 15, 2011.
- Jacco van Ossenbruggen, Michiel Hildebrand and Victor de Boer. Interactive vocabulary alignment. Proceedings of the International Conference on Theory and Practice of Digital Libraries 2011. Berlin, September 26–28, 2011.