OpenSKOS next edition: triplestore support for controlled vocabularies

Olha Shkaravska
Technical Development
Meertens Institute,
The Netherlands
olha.shkaravska@meertens.knaw.nl

Menzo Windhouwer
Technical Development
Meertens Institute,
The Netherlands
menzo.windhouwer@meertens.knaw.nl

Marc Kemps-Snijders
Technical Development
Meertens Institute,
The Netherlands
marc.kemps.snijders@meertens.knaw.nl

Abstract

OpenSKOS software implements a web-service-based approach to management and use of vocabularies based on SKOS design principles. This paper motivates and describes the most recent developments in the software. These developments include migrating from a relational MySQL database to a triplestore and corresponding changes in the software. Also, relation management has been extended so that user-defined relations can be added and used along with SKOS-specified relations. This transfer ensures smooth upgrade of the OpenSKOS-based projects and reusing those parts of the software that have preserved their relevance.

1 Introduction

OpenSKOS software implements a web-service-based approach to publication, management and use of vocabularies based on SKOS design principles. SKOS stands for Simple Knowledge Organisation System and is a series of RDF-based recommendations by W3C for maintaining entries in structured controlled vocabularies. These recommendations are meant to support standardization and interoperability in storing and exchange of digitalized data. Moreover they comprise the best practices and experience in the area. For these reasons many digital-humanities and administrative vocabularies are run on SKOS supporting platforms.

As it follows from its name, OpenSKOS is one of those platforms. The name refers to "infrastructure and services to provide *Open Access* to SKOS data." It has been originally developed in the Netherlands "CATCHPlus" project[1] and now is used by various Netherlands cultural heritage institutes. Meertens Institute has joined them to collectively use, maintain and further develop the platform. At present two of CLARIN registries hosted at Meertens Institute – CCR (CLARIN Concept Registry, [Schuurman et al. 2016]) and the CLAVAS (CLARIN Vocabulary and Alignment Service, [Brugman, 2016]) - are run on OpenSKOS software. OpenSKOS is subject to the GNU General Public License version 3.

This paper motivates and describes the most recent developments in the OpenSKOS platform. These developments include migrating from a relational MySQL database to a **triplestore** and corresponding upgrades in the architecture of the software. Also, relation management has been extended so that user-defined relations can be added and used along with SKOS-specified relations, like *narrower* or *broader,* between concepts. The most important aspects of the earlier versions of OpenSKOS, which are transferred to the current one, like API signatures, are discussed here as well. This transfer ensures smooth upgrade of the OpenSKOS-based projects and reusing those parts of the software that have preserved their relevance.

[1] http://www.catchplus.nl

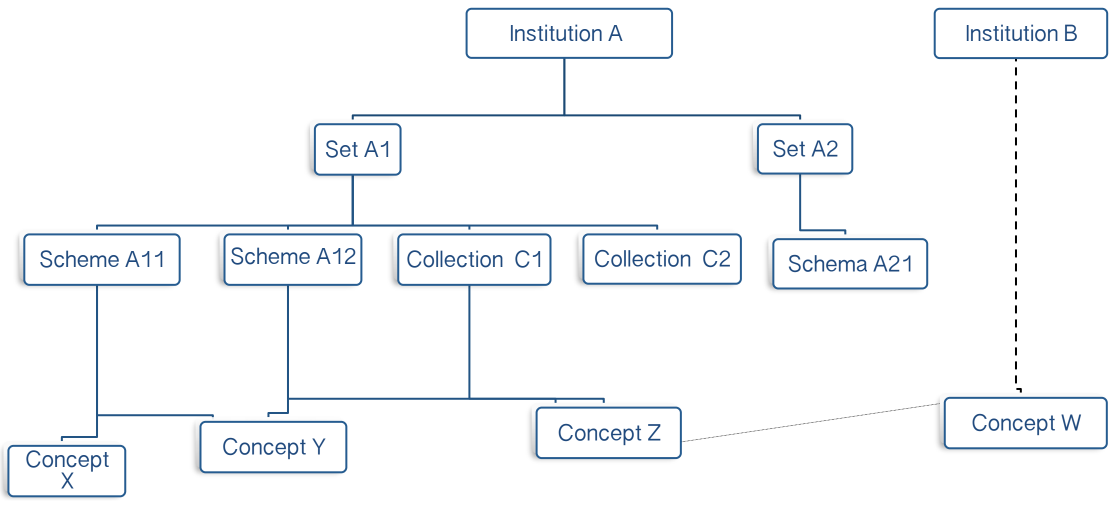

Figure 1. OpenSKOS data model

Behind the implementation of the OpenSKOS there is a resource model based on 6 types of resources: *institution*, *set* (formerly known as *institution collection*), *concept scheme*, *collection*, *concept* and *relation*. In *Figure 1 Concept Z* and *Concept W* are connected via a SKOS - or a user - defined relation. Sets and institutions are not part of the SKOS model but added for practical reasons. Sets can group a number of concept schemes, concepts and collections together that constitute one resource from an organizational/data management perspective. Sets also play the role of OAI-PMH sets used for selective harvesting. Institutions are added to make information available on the vocabulary publishers themselves, and to associate authorized vocabulary managers with. Different institutions cannot share sets.

2 RESTful API

The original system's API is defined in a collaborative effort between the CATCHPlus project office, three major commercial tool providers for the Netherlands Cultural Heritage sector (Adlib Systems, Picturae and Tresorix) and the Cultural Heritage Agency of the Netherlands. The API of the preceding version allowed to search for SKOS concepts, concept schemata ('vocabularies'), institution collections and institutions by sending REST requests to the backend and to obtain response in a number of representation formats (JSON, RDF/XML, HTML). Query parameters enabled filtering on properties relevant for a given resource type, and specification of what information is/is not included in the result.

The current API has completely inherited the previous interface for concepts and concept schemata. The API for institution collections is renamed to the API for sets, and the API for institutions lost plural 's' at the end for consistency. The API for collections and relations has been added.

As in the previous version, the API has 'find' functionality for concepts. It supports a query parameter 'q' that takes queries according to the Apache Lucene Query Parser Syntax as values. Searching is possible over all SKOS based fields and over Dublin Core (*dcterms*) fields, if those are present. The result of a 'find' query is a list of concepts (represented in the same way as for the concept resolve) and a diagnostics block, for example with number of results that match and number of results on page. Paging and sorting of results is supported. A specialization of the find API is the OpenSKOS 'auto complete' function, meant for interactive searching for matching concept labels starting with some characters.

The OpenSKOS API not only supports HTTP GET actions, but it also supports PUT, POST and DELETE operations for all types of resources. It is therefore possible to perform vocabulary maintenance tasks directly on the repository using the API. For REST examples see openskos.org.

Picturae has implemented the new OpenSKOS triplestore-oriented functionality for the already existing API, while Meertens Institute has implemented functionality behind the extended API. For that purpose the original code has been significantly refactored. Moreover at Meertens Institute a completely new concept browser has been designed, which uses the Nederlab[2] front-end architecture. The browser uses the upgraded API and not only implements the functionality of the previous CCR browser, but also implements graphical and table views for relations.

3 Triplestore advantages for OpenSKOS

Another name for a triplestore database is an **RDF store**, and both names speak for themselves. This is a database for storing, updating and retrieving triples of the form (*subject*, *property*, *object*). For instance, a SKOS concept within such a database is represented via a series of triples of the form (*concept's uri*, *property name*, *property value*). A property name is given by an URI, which has a corresponding short version *skos:prefLabel*. The value of a property in the triplestore can be either a reference (the URI of another resource), or a literal. For example, the value of *skos*:*inScheme* is a reference to a concept scheme, and *skos:prefLabel* is a literal with a language attribute.

What are the reasons behind the decision to move concepts, schemata and other related resources from the relational MySQL storage to the triplestore? First of all it is clear that if one represents data within an RDF-based model, such as SKOS, then it is natural to store these data within a database that has been designed upon the RDF model, and this database is the triplestore. The other reasons, such as maintaining and extension possibilities, are as a rule, strongly connected with this theoretical observation.

Secondly, storing RDF in MySQL tables leads to technical overhead in support of OpenSKOS platform, since one has to maintain translations from RDF structures to their relational representations and vice versa.

Thirdly, the triplestore seems a reasonable choice from the point of view of possible extensions of the model. Implementing such extensions in a relational database amounts to altering the old relational model and to significant changes in the software layers behind the API layer. At the same time the triplestore, when adding new kinds of triples, is not altered at all, and a good written intermediate software layers can be so generic that the changes there will be minimal. We believe that this is the case for the new implementation of OpenSKOS.

The fourth reason can be regarded as an illustration for the previous case. Within CLARIN it was decided to extend model with relations between concepts. In fact, SKOS already has some build-in relations like *broader* and *narrower* between concepts or *hasTopConcept* between schemata and concepts. It does make sense though to support user-defined relations. The definitions of such relations can be considered as a regular resource and maintained as a series of triples like any other resource. Thus, to add, update and retrieve a relation description is the same as to perform these actions on a concept or a schema. Except that contrary to the other types a relation does not have *openskos:uuid* property and its URI (*rdf:about* property) must contain the prefix for the namespace and its short name in the spirit of SKOS design. An example of a relation description in the XML format looks as follows (*dcterms:datesubmitted* is omitted and URI of the creator is shortened for presentation purposes):

```
<rdf:Description rdf:about="http://somewhere.org/xmlns#better">
   <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#objectProperty"/>
   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2004/02/skos/core#related"/>
   <dcterms:creator rdf:resource="http://somewhere/users/9c34ded0"/>
   <dcterms:title>better</dcterms:title>
</rdf:Description>
```

One may need to add derivation rules to a dedicated *rules*-file for the triplestore instance under consideration, e.g. to reflect transitivity or the fact that two relations are inverses of each other. At the moment to assure removal of the triples of the form (*concept-A*, *relation name*, *concept-B*) after *concept-A* or *concept-B* is removed, pairs of mutually-inverse-relation names must be placed in a configuration file as well.

Last but not least. The chosen Jena/Fuseki implementation of the triplestore uses the Simple Protocol and RDF Query Language (SPARQL), which is made a standard by the RDF Data Access Working Group and is one of the key technologies of semantic web. As well as SKOS model,

[2] https://www.nederlab.nl

SPARQL endpoints become more and more popular amongst different research and governmental institutions. The OpenSKOS working group has made a decision to follow this general trend. Due to interoperability of SKOS and SPARQL, this widens opportunities for collaboration with other institutions and companies.

4 New OpenSKOS backend implementation for triplestore

One can single out 4 main layers of OpenSKOS software that support data flow between API REST requests and the triplestore: (1) API controllers, (2) the deepest layer of 6 resource managers which request and update the triplestore, (3) the intermediate layer between the deepest layer and the controllers, (4) a resource-description layer implementing 6 resource types as classes, and a validation module.

The key technical point behind the implementation of these layers is that 6 different types of data are treated in a generic way as a ‘resource’ because have the same nature and often share common properties. On each layer there is one (abstract) parent class implementing common functionality. Specific for a concrete resource type procedures are implemented in the corresponding concrete child class. For instance, while posting a resource, not all properties can be present in the request xml body. Some properties like the resource creator, or the date-time of creation/update can or must be derived from the request. These properties differ per resource and therefore their handling is implemented in a specific *addMetada* procedure on resource description level in the corresponding resource type implementations. Still, currently collections and schemata share the same procedure that mainly reuses the corresponding parent one.

The validation module is the least generic part of OpenSKOS software. This is not surprising because different resources have different properties. Nevertheless, there is still significant code share there because the validation algorithm is generic and the properties of different resource types often overlap, e.g. both *institutions* and *sets*, have a unique property *openskos:code*. Validators are activated when a request to store or to update a resource is sent. They evaluate if the sent XML description is valid with regard to a given type of resource. Validation is two-fold: firstly, it must check if obligatory fields are present (and single if applicable); secondly, property values are checked. Reference values are valid if the referred resources of the corresponding types exist in the triple store. Unique literal values like the values of *openskos:code* or *openskos:uuid*, are, of course, checked for uniqueness.

There are two more software modules worth mentioning: *Easyrdf* class, which turns triplestore responses on SPARQL requests into a (collection of) resource(s) whose type is passed as a parameter, and *MyInstitutionModule* where authorization and URI generation procedures are implemented. The institution should be able to easily adjust the implemented approaches by altering code in this module according to the institution demands.

At the time of writing of this article the software has been under testing of its functionality and performance. Earlier performance tests had shown that *full text search* of resources in the triple store was very slow. For this reason the decision was made to include an intermediate indexing layer, namely a SOLR/Lucene index for concepts, and the corresponding software modules that handle connection to this index and its synchronization with the triple store were added. After adding the SOLR/Lucene layer search has become scalable. Performance of *update* operations is evaluated by runs of import and migration scripts over large amount of resources. Import means that the concepts (and possibly other resources) represented in an XML file must be transferred to the triple store. Migration means that the resources (e.g. schemes and concepts) represented as SOLR documents must be transferred to the triple store and another SOLR/Lucene index with upgraded field naming. Both, import and migration, include validation of the resources to be transferred. A non valid resource is not transferred to the new store and the corresponding error message gives detailed information on the reason for it, so that the user can correct the resource’s representation. At present the idea about the performance of update operations can be given by the following figures:

- import 10,000 concepts (all valid, without relations) takes 970 seconds, run on Ubuntu 14.04.5 Docker version 1.8.1;
- migration 3,205 documents (16 schemes and 2,010 valid concepts, no relations) takes 1,292 seconds, run on Ubuntu 14.04.1 LTS in the Docker version 1.6.2.

Note that the update performance of the new version of OpenSKOS is worse than performance of its current version which is a consequence of the fact that validation, like checking references or uniqueness of certain elements, has become way more thorough than in the earlier implementations.

5 Conclusions and future work

A new OpenSKOS platform for communication with the triplestore database has been implemented and its API has been extended with a number of functionalities. By the time of writing this article CCR and CLAVAS have been migrated to the new system on a testing server. The first testing results, including performance and functionality, look promising. The new backend can be used in read-only mode in the nearest future, but the editor implementation still needs to be upgraded according to the updates in API and the architecture. Still, sending POST and PUT requests works as direct HTTP requests to the server. The implementation of relations can be extended and improved after the first feedback is obtained and certain experience is accumulated. The current implementation still uses an Apache SOLR layer, which may be replaced in the future by other search-facilitating engine. In any case, the indexing layer is necessary for acceptable performance of full text search.

As it has been mentioned in the beginning on this article, many digital-humanities and administrative vocabularies are run on SKOS supporting platforms. Some of these platforms are open source, like Opentheso, whereas other powerful ones, like Intelligent Topic Manager of Paris-based “Mondeca”, are commercial. However, due to the involvement of CLARIN centers OpenSKOS can closely follow the needs of the CLARIN infrastructure, and is also backed up by a wide spectrum of users-institutions. Still it’s important to compare the functionality and performance of OpenSKOS with the similar products and initiatives, to prevent lock-in but also to see where non-standardized functionality, e.g. the SKOS-based REST API, can converge.

Acknowledgements

The authors like to thank the colleagues from OpenSKOS expert panel – Picturae, The Netherlands Institute for Sound and Vision, Cultural Heritage Agency of the Netherlands - for the fruitful discussions on the experiences around the OpenSKOS. They also like to thank CLARIN management and colleagues for support in developing and testing the software. Part of this work was funded by the CLARIN-PLUS project.

References

[Schuurman et al. 2016] I. Schuurman, M. Windhouwer, O. Ohren, D. Zeman. *CLARIN Concept Registry: The New Semantic Registry*. In K. De Smedt (ed.), Selected Papers from the CLARIN 2015 Conference, Linköping Electronic Conference Proceedings, April, 2016.

[Brugman, 2016] H. Brugman. *CLAVAS: a CLARIN Vocabulary And Alignment Service*. In J. Odijk and A. van Hessen (eds.), CLARIN in the Low Countries, Ubiquity Press, 2016. (Upcoming)