D2.6

Reinforced concept registry (software)

Document information

Title	Reinforced concept registry (software)
ID	CLARINPLUS-D2.6 (CE-2016-0927)
Author(s)	Olha Shkaravska, Menzo Windhouwer, Marc Kemps-Snijders
Responsible WP leader	Dieter van Uytvanck
Contractual Delivery Date	2016-12-31
Actual Delivery Date	2016-12-30
Distribution	Public
Document status in workplan	Deliverable

Project information

Project name	CLARIN-PLUS
Project number	676529
Call	H2020-INFRADEV-1-2015-1
Duration	2015-09-01 – 2017-08-31
Website	www.clarin.eu
Contact address	contact-clarinplus@clarin.eu

Table of contents

1 Executive Summary

The 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[1] for maintaining entries in structured controlled vocabularies. These recommendations are meant to support standardization and interoperability in storing and exchanging digitized 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 "CATCHPlus" project[2] and is now used by various Dutch cultural heritage institutes. The Meertens Institute has joined them to collectively use, maintain and develop the platform. At present two CLARIN registries hosted at the Meertens Institute – CCR (CLARIN Concept Registry, [Schuurman et al. 2016]) and CLAVAS (CLARIN Vocabulary and Alignment Service, [Brugman 2016]) - are based on OpenSKOS software. It is subject to the GNU General Public License version 3.

The OpenSKOS framework consists of two main ingredients: the *backend* that includes the database and the software to access and manage it, and the *frontend* that provides convenient user interface for searching and changing the database. In its turn, the frontend software consists of two parts: the *browser* and the *editor*. The browser facilitates the search within a chosen database for concepts and relations between them. Authorisation is not needed. The browser software is fully separated from the backend software. The editor (still under development) allows to add and to update OpenSKOS resources, for this authentication and authorization is needed. At the moment the editor software is a submodule of the backend code. The sources of the backend software can be found at https://github.com/OpenSKOS/OpenSKOS. The software maintained by the Meertens Institute can be found at "developMeertens" branch.[3] The sources for the browser can be found at https://github.com/meertensinstituut/OpenSKOS-browser. For both ingredients one can find instructions on usage and installations in the included Readme documents.

The new beta versions of the OpenSKOS-based CCR and CLAVAS are available via https://www.clarin.eu/openskos

[1] World Wide Web Consortium (W3C), see https://www.w3.org/
[2] See http://www.catchplus.nl
[2] See http://www.catchplus.nl
[3] A merge of the developMeertens branch into the master branch is planned to happen after the first official of OpenSKOS 2.0, which is under active development of Picturae and the Institute of Sound and Vision.

2 Introduction

This document reports on the implementation of Subtask T 2.2.3 – "Reinforced concept registry [CLARIN]" by the Meertens Institute in Amsterdam[4]. The background behind this task is as follows. Until the end of 2014 CLARIN used a separate Data Category Registry (ISOcat) and a semi-controlled vocabulary registry (CLAVAS based on OpenSKOS). As these services partially overlap in functionality and as the ISOcat organizational future had changed (the Max Planck Institute for Psycholinguistics stopped being the DCR ISO Registration Authority in December 2014), the decision has been made to migrate the data categories used in CLARIN context to an OpenSKOS registry.

Basic changes were made to OpenSKOS to make it a suitable replacement of ISOcat, and the data categories used by CLARIN have been migrated from ISOcat to the new CLARIN Concept Registry (CCR). But more changes were needed to better fit OpenSKOS into CLARIN's metadata workflow (CMDI), e.g., relations among concepts, easier import of new concepts, concept lifecycle management and browse and search facilities for unauthenticated users. These changes have been discussed and aligned with the OpenSKOS community to leverage the benefits of shared development. Besides the Meertens Institute this community includes the Netherlands Institute of Sound and Vision, the Cultural Heritage Agency of the Netherlands, the commercial company Picturae and the Austrian Centre for Digital Humanities. Based on the outcome of these discussions Picturae has developed the software for managing the triple store database with concepts. The Meertens Institute has been adjusting this software further for CLARIN purposes. Moreover, the Meertens team has developed a new browser compatible with the upgraded database and the software behind it, i.e., the backend.

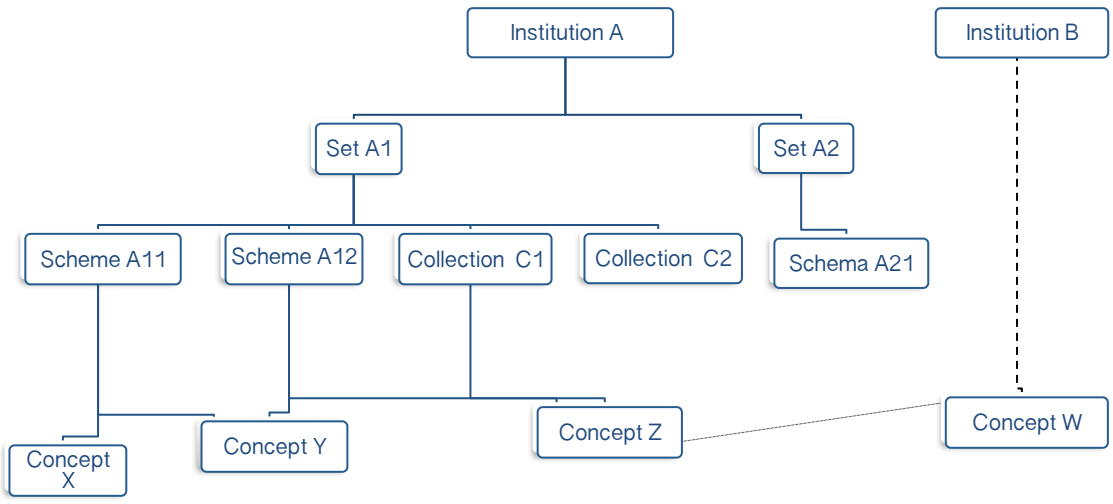

Figure 1. OpenSKOS data model.

Behind the implementation of 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

[4] https://office.clarin.eu/v/CE-2015-0688-CLARIN-PLUS_CCR_analysis.pdf

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.

This resource model and the task objectives motivate the choice of the triple store as the storage for OpenSKOS resources [Shkaravska2016]. It is explained in details in the section 3.1, and the details of the implementation are given in section 3.2. Changes in the management of the database and introducing new features imply also changing the API. Basically existing API calls stayed intact, but a number of new services have been added. The API is discussed in section 4. Section 5 is devoted to the browser software and section 6 concludes this document.

3 Triple store for concept registries

3.1 Advantages of a triple store

Another name for a triple store is an **RDF store**. 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, e.g., *skos:prefLabel*. The value of a property in the triple store 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 database, which stored resources in the previous version of the OpenSKOS, to the current triple store? 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 this data within a database that has been designed upon the RDF model, and such a database is a triple store. 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 the OpenSKOS platform, since one has to maintain translations from RDF structures to their relational representations and vice versa.

Thirdly, the triple store 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 triple store, when adding new kinds of triples, is not altered at all, and 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 the 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. However, it does make sense though to also support user-defined relations. The definitions of such relations can be considered a regular resource and maintained as a series of triples like any other resource in the triple store. 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 RDF/XML format looks as follows (*dcterms:dateSubmitted* is omitted and URI of the creator is shortened for presentation purposes):

```
<rdf:Description rdf:about="http://menzo.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>
```

The snippet above is the body of the response on the GET request

`/public/api/relation?id=http://menzo.org/xmlns%23better`.

This body represents a user-defined relation with a title "better" and the URI http://menzo.org/xmlns#better. The URI on itself does not have to be resolvable, that is `http://menzo.org/xmlns%23better` does not have to redirect to an existing page, but a GET request with the "id" parameter set to the URI yields the corresponding resource or 404 if the resource does not exist. A URI must consist of two parts: the namespace URI and the actual name separated by the hash symbol. When POSTing a new relation definition the user has to supply the relation's name and the URI in the request body. The elements "type", "subpropertyOf", "dateSubmitted" and "creator" are derived from the API type "relation" and the authentication information, tenant and user key, supplied as request parameters.

One may need to add derivation rules to a dedicated *rules*-file for the triple store 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[5] of the triple store supports the Simple Protocol and RDF Query Language (SPARQL). This is a standard made by the RDF Data Access Working Group and is one of the key technologies of the semantic web. Like the SKOS model, 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.

3.2 Implementation

One can single out 4 main layers of OpenSKOS software that support data flow between API REST requests and the triple store.

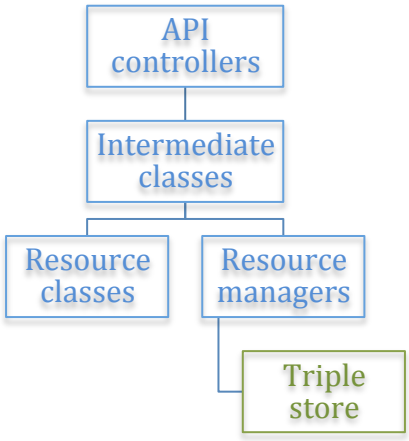

Figure 2. Backend structure with 4 software layers (blue).

[5] See https://jena.apache.org/

API calls amount to the calls to the corresponding API controllers that communicate with the triple store via the deepest software layer consisting of 6 resource managers. The resource managers send queries and update the database. They communicate with the API controllers via intermediate classes and also use resource description layer implementing 6 resource types as classes.

The key technical point behind the implementation of these layers is that 6 different types of data (see section 2) 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 are derived from the request. These properties differ per resource and therefore their handling is implemented in a specific *add Metadata* 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 if they are single when applicable (e.g. *openskos:set* is single in a concept schema description, because a schema cannot belong to multiple sets, however many different concept schemata may refer to the same set, so the value of *openskos:set* element is not unique). Secondly, property values are checked as well. 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 triple store 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.

The software has undergone 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 the 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. An invalid 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:

- the import of 7679 concepts (all valid, without relations) takes 575 seconds, run on Ubuntu 14.04 with Docker version 1.8.1;
- the migration of 3205 documents (16 schemes and 2010 valid concepts, no relations) takes 1292 seconds, run on Ubuntu 14.04 with 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.

4 RESTful API

The original system's API was defined in a collaborative effort between the CATCHPlus project office, three major commercial tool providers for the Dutch 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.

4.1 Implementation

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 GET request of api/concept) and a diagnostics block, for example with the number of matching results. 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 http://www.openskos.org/api.

Picturae has implemented the new OpenSKOS triple store-oriented functionality for the already existing API, while the Meertens Institute has implemented functionality behind the extended API. For that purpose the original code has been significantly refactored.

5 New concept registry browser

When the first versions of the new backend software were delivered and the corresponding upgraded API became available, it became clear that upgrading of the frontend software, such as the browser and the editor, is inevitable, e.g. adding the functionality for handling relations via a convenient user interface. To make development and further support of the upgraded browser within the Meertens institute more effective it has been decided to redesign the browser according to the architecture of the frontend of the Nederlab project [Brugman 2015] launched by the Meertens Institute in the begin of 2015. This allows reuse of the corresponding software and the expertise.

5.1 Smarty templates-based implementation

Browsing software is completely separated from the OpenSKOS backend software. It uses the Smarty PHP library for generating HTML templates and filling their placeholders with the data via server-side variables (PHP) or the client-side renderer (JavaScript). Both sorts of data, input request parameters defined by the user's choice, and the server response as output data, can fill-in pre-generated Smarty templates when the application runs. The HTML forms filled by the user's selections define a search request, for instance search for concepts with a certain word in its preferred label.

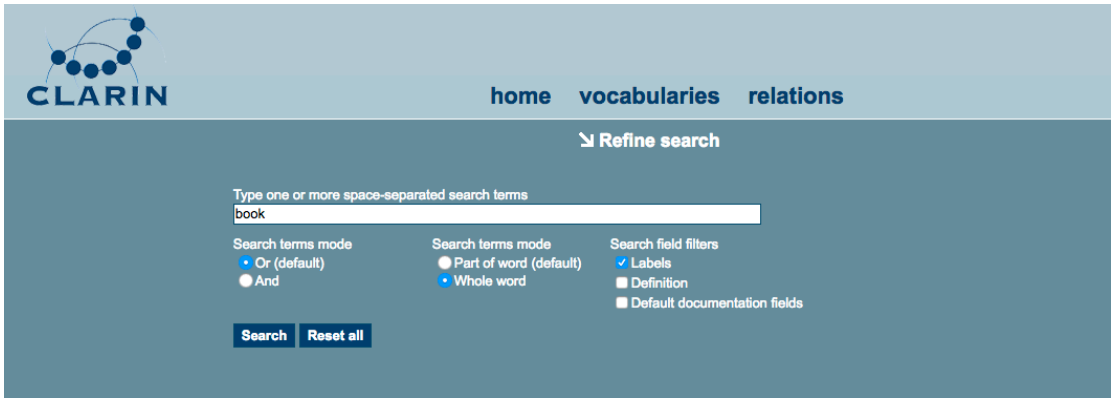

Figure 3. Example of the refinment of search in concepts (vocabularies): string "book" should occur in all three sorts of labels (preferrable, alternative and hidden) as a separate word.

As in the previous browser, one can filter search by concept status, schema, collection and institution. The database, CCR or CLAVAS (or any other OpenSKOS-API-compatible database included in the configuration), can be chosen on the home page of the browser. Note, that the previous version of the browser was designed only for CCR, and CLAVAS did not have its publically available UI for searching within the concept database.

After a search form is submitted, the PHP index page invokes a number of PHP helper methods, which transfer the request parameters to the JavaScript controller. In its turn it triggers the AJAX request to the database. The backend's response is overworked by JavaScripts that fill in the HTML page with the response data in the view part of the package. Note that a proxy (PHP) is needed to send AJAX requests to "another", i.e., the backend's domain.

An option for browsing relations is implemented in the new browser. On can filter the search by the name of the relation, its source and target schemata.

Filters

Relations
skos:broader ✓
skos:narrower
skos:related
skos:broaderTransitive ✓
skos:narrowerTransitive
skos:broadMatch
skos:narrowMatch
skos:closeMatch
skos:exactMatch
skos:relatedMatch

Source schemata
ISO 639-3
OpenSKOS-CLARIN Organizations

Target schemata
ISO 639-3
OpenSKOS-CLARIN Organizations

Found 353 relations

Subject	Subject schema	Property	Object	Object schema
Department of Scandinavian languages and literature, University of Helsinki	OpenSKOS-CLARIN Organizations	http://www.w3.org/2004/02/skos/core#broader	University of Helsinki	OpenSKOS-CLARIN Organizations
Department of Modern Languages, University of Helsinki	OpenSKOS-CLARIN Organizations	http://www.w3.org/2004/02/skos/core#broader	University of Helsinki	OpenSKOS-CLARIN Organizations
Institute of Development studies, University of Helsinki	OpenSKOS-CLARIN Organizations	http://www.w3.org/2004/02/skos/core#broader	University of Helsinki	OpenSKOS-CLARIN Organizations
Department of Classical Philology, University of Helsinki	OpenSKOS-CLARIN Organizations	http://www.w3.org/2004/02/skos/core#broader	University of Helsinki	OpenSKOS-CLARIN Organizations
Department of General Linguistics, University of Helsinki	OpenSKOS-CLARIN Organizations	http://www.w3.org/2004/02/skos/core#broader	University of Helsinki	OpenSKOS-CLARIN Organizations

Figure 4. Response on the request for all skos:broader and skos:broaderTransitive relation triples, in the form of a table (fragment).

The server response on the relations search request can be represented either in the form of a table or in the form of a graph.

Figure 5. Response on the request for all skos:broader and skos:broaderTransitive relations, in the form of a graph (fragment).

6 Conclusion

A new OpenSKOS platform based on a triple store database has been implemented and its API has been extended with a number of functionalities. By the time of writing CCR and CLAVAS have been migrated to the new system on a testing server and the production migration of CLAVAS to OpenSKOS 2 is underway. The first testing results, including performance and functionality, look promising. As a next step, the possibility of using non-exhaustive vocabularies during the creation and curation of metadata records with CLAVAS will be presented to the CLARIN centres. During a limited-scale pilot phase the use of CLAVAS in this context will be evaluated.

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 works, so API-based maintenance is very well possible. 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 another search-facilitating engine. Currently the indexing layer is necessary for acceptable performance of full text search.

As it has been mentioned before, many digital humanities and administrative vocabularies are run on SKOS-supporting platforms. Some of these platforms are open source, like Opentheso[6], whereas other powerful ones, like Intelligent Topic Manager[7], are commercial. However, thanks to the involvement of various CLARIN centers with the OpenSKOS community, OpenSKOS can closely follow the needs of the CLARIN infrastructure, and is also backed up by a wide spectrum of users-institutions.

Acknowledgements

The authors like to thank the colleagues from OpenSKOS expert panel – Picturae, the Netherlands Institute for Sound and Vision, the Cultural Heritage Agency of the Netherlands - for the fruitful discussions on the experiences with OpenSKOS. They also like to thank the CLARIN colleagues for support with developing and testing the software.

[6] See https://github.com/miledrousset/opentheso

[7] See http://www.mondeca.com/137-2_trashed/itm/

References

[Shkaravska 2016] O. Shkaravska, M. Windhouwer and M. Kemps-Snijders, *OpenSKOS next edition: triplestore support for controlled vocabularies*, In proceedings of the CLARIN Annual Conference 2016, https://office.clarin.eu/v/CE-2016-0917-Proceedings-CAC-2016.pdf .

[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)

[Brugman 2015] H. Brugman, N. van der Sijs, R. van Stipriaan, E. Tjong Kim Sang, A. van den Bosch, J. P. Kunst, R. Zeeman, D. Kooij, I. Brussee, M. Brouwer, M. Kemps-Snijders en H. Bennis. Nederlab: Towards a single portal and research environment for diachronic Dutch text corpora, in proceedings of the Tenth International Conference on Language Resources and Evaluation, LREC 2016. (http://www.lrec-conf.org/proceedings/lrec2016/index.html)