

└ SWAD-Europe Thesaurus Activity

Deliverable 8.7

RDF Thesaurus Prototype

Abstract:

This report describes the thesarus research prototype demonstrating the SKOS schema by means of the SKOS API web service and a demonstrator containing sample data, some simple clients for using the API, documentation and description of related work.

Project name:

Semantic Web Advanced Development for Europe
(SWAD-Europe)

Project Number:

IST-2001-34732

Workpackage name:

8. Thesaurus Research Prototype

Workpackage description:

└

<http://www.w3.org/2001/sw/Europe/plan/workpackages/live/esw-wp-8.html>

Deliverable title:

8.7: rdf_thesaurus_prototype

This version:

└ <http://www.w3.org/2001/sw/Europe/reports/thes/8.7/>

Latest version:

└ <http://www.w3.org/2001/sw/Europe/reports/thes/8.7/>

Authors:

Dave Beckett, ILRT, University of Bristol
Nikki Rogers, ILRT, University of Bristol



Status of this document

Completed 2004-09-28.

This document describes work on the prototype for the RDF Thesaurus service, consisting of a web service API described in WSDL, an example server-side implementation of the web service API, written in Java and using Sesame, and a client written in Python to invoke the web services along with live demonstrations of the server, client and web service API using sample data.

Comments on this document are welcome and should be sent to the authors or to the [└ public-esw-thes@w3.org](mailto:public-esw-thes@w3.org) list. An archive of this list is available at [└ http://lists.w3.org/Archives/Public/public-esw-thes/](http://lists.w3.org/Archives/Public/public-esw-thes/).

Contents

1. [Introduction](#)
 2. [SKOS API](#)
 3. [SKOS Thesaurus Demonstrator](#)
 - 3.1 [DREFT Thesaurus Web Service Server](#)
 - 3.2 [DREFT Thesaurus Web Service Client](#)
 4. [Issues](#)
 - [References](#)
-

1. Introduction [\[back to contents \]](#)

The SKOS-Core (as described in the [Introduction to SKOS-Core 1.0 Guide](#)) describes a data model for thesauri or terminologies that enables them to be described in semantic web terms, using RDF. This provides a data-level description but does not deal with how web based applications can use such data. This deliverable describes the development of an API to the SKOS-Core, implemented using standard web service technologies such as XML, SOAP and WSDL. The API methods (the web service operations) follow on from the [use cases](#) we have produced, which themselves are drawn from input from the existing thesauri/KOS user community and experience drawn from previous applications in this area.

2. SKOS API

2.1. Introduction and Goals -

The SKOS API defines a web service providing a core set of operations for accessing and querying a thesaurus or terminological resource based on the SKOS-Core schema, as described in the [Introduction to SKOS-Core 1.0 Guide](#).

Our main goal has been to initiate the on-going development of a web service API that will be suitable for widespread adoption, which in turn will promote ease of interoperability and re-use of information systems that exploit thesauri and/or other kinds of terminological resource.

2.2. Approach -

To meet our objectives we have developed an API based on the concepts in the SKOS-Core. This API is intended for use by terminology service clients or as the basis of new services. As noted above, the approach taken has been directed by [use cases](#) that existing users of thesaurus services described to us as the fundamental functions needed by their applications. From these use cases we have generated the specific access methods needed for a service that provides access to a SKOS-Core encoded thesaurus. This is described in the [SKOS-Core RDF schema](#). We made a decision to base the API entirely around the terminology of the SKOS-Core application space, rather than in purely RDF/Semantic Web terminology; in other words we have used *Concepts*, *Relations* and *Mapping* etc. as method parameters, rather than RDF resources, properties and so on, although these are still used.

Our intended scenario has been for the API to be delivered as a Web Service, using the standard mechanisms and formats of SOAP and WSDL to describe it, with a demonstrator web service access point over a data store of some sample thesauri encoded in the SKOS-Core schema. To demonstrate Web Service consumption in a

distributed environment, we set an objective to write a client to use the web service preferably in a different implementation language than that of the demonstration server, thus enforcing the use of the web service API rather than operating more of a "back door" route, such as via a language or platform-specific API.

2.3. Identification -

One key issue in the API, and of SKOS-Core is that of identification of the key items of interest:

Item	In SKOS
<i>Concept</i>	Class <code>skos:Concept</code>
<i>Relation</i>	Property <code>skos:semanticRelation</code>
<i>Thesaurus</i>	Class <code>skos:ConceptSchema</code>
<i>Top Concept</i>	Class <code>skos:Concept</code> and <code>skos:TopConcept</code> Although this will likely change to a property <code>hasTopConcept</code> of the Thesaurus.

Thesauri are always identified by URIs as they are larger items that should have a global name for ease of use.

The SKOS-Core guide describes how concepts are either labeled by a URI where they have one, or alternatively have an `external non-URI identifier` which is actually an identifier local or scoped to some thesauri (This term may change to `skos:localID` in a future revision of the schema). These two items mean that the API calls could potentially be doubled for each method that needs to ask about a *Concept* - `GetXYZByURI` and `GetXYZByExternalID`. Instead, we chose to use the *Concept* object itself so the URIs and externalIDs become fields of the *Concept* object (`Concept javadoc`) Thus to make a query about a *Concept*, the appropriate fields of a *Concept* object are filled in (URI or externalID) and the method invoked with the *Concept* instance given as the parameter.

Relations are always identified by URIs. (`Relation javadoc`)

2.4. Supporting Browsing and Searching -

Development of the API has been directed toward clients that do the two most common styles of interaction with KOS data - the browsing and searching of terminological information.

Browsing needs navigating up, down and around the concepts (BT, NT, etc as discussed the guide) which means being able to navigate from the starting position of a known concept. Browsing also needs entry points to start from and these can include the top concepts, if indeed the thesaurus has some.

Searching requires being able to give some form of free text keyword or regular expression (regex) and get back some matching concepts. Both of these require read-only access to the information, which simplifies the problem considerably.

The information displayed during browsing can be quite detailed or very brief such as one descriptive field, but that is more of an application issue than something for the API level at this stage. Rather than have multiple information-depth interfaces, we chose to return a *Concept* structure that contains the most useful SKOS-Core terms:

- `skos:definition`
- `skos:example`
- `skos:altLabel`
- `skos:prefLabel`
- `skos:scopeNote`

2.5. Terminology and Methods -

As previously described the terminology used is from SKOS-Core - *Concept*, *Thesaurus* and *Relation*. Thesauri are only ever referenced as a parameter in the use cases so, since they are identified by a URI, these need no specific concept in the API. This reduces the classes needed to ones for *Concept* (`Concept javadoc`) and *Relation*

([⌊ javadoc](#)), in the simplest API. We decided to exclude concept mapping for the first version of the API. The introduction of concept mapping support to the API would mean the addition of a Mapping class.

Our first objective was to create a WSDL file describing a web service. This is a complex XML file written using W3C XML Schema for describing the operations and types used by the service, as well as the interaction styles. The file is primarily intended for machine-to-machine (m2m) usage and typically is generated by the SOAP software used. We opted to use the RPC/Encoded service style for this first version of the Web Service implementation - our decision being largely influenced by the relative availability of open source tool support for RPC/Encoded (ahead of Document/Literal) at the time the workpackage commenced. The software we adopted for SOAP support - [⌊ Apache Axis](#) - automatically handles the SOAP encoding of java beans (we created beans for each of Concept, Relation and URI for example) - in other words it handles the mapping from Java objects to XML for SOAP message content, and vice versa.

The general lifecycle pattern we used therefore with Axis was first to auto-generate the WSDL file - and this was generated from our definition of the API as a series of well-documented Java interface classes. We used the Apache Axis [⌊ Java2WSDL](#) tool for this purpose (making the XML for the software). This has the additional benefits of generating readable JavaDoc, providing a handy reference to the API, as well as allowing Java to validate the interface syntax used. The Java2WSDL tool had some limitations but this has proved a good approach. From the WSDL we then generated client-side stubs (using the complementary Axis WSDL2Java tool) which we then used to support the initial testing and development of the Web Service software.

The service's WSDL file needs to be applicable to any implementation language, and it was seen as best practice to not overload API methods, since some languages do not support method overloading. Thus where API methods differ only in the types of their parameter, the method name gains a suffix such as *ByThesaurus* or *ByExternalID*. We are also aware that the best practice *WS-I Basic Profile* [⌊ \[WSIBP\]](#) recommends against method overloading.

The method responses are most often sets of answers, sometimes ordered, so a Java array has been used as this maps correctly into the WSDL as an XSD complexType, array of item. The items returned can be relatively heavyweight if they are a sequence of serialised Concept objects. This is a trade off where the size of the response is large, but the alternative of making multiple request/response calls to find out more detail may be even more costly. This is something that can only be found out by implementation.

We have kept the API minimal rather than having lots of fine grained calls for returning information via various classes such as, say, *GetPreferredTermsOfConcept*. So any method returning a Concept (such as 'getConcept') responds with all the Concept's fields in one call. Since the SKOS-Core schema properties for a Concept are relatively few and there are a small number of optional properties, this has seemed a good approach.

Two API methods have required an additional class to return the results, since the return types are limited to objects or arrays of objects. These methods were those returning concepts at a "path" distance from a specified concept: *getAllConceptsByPath* and *getConceptRelativesByPath*. These required the definition of a new class *ConceptRelatives* ([⌊ javadoc](#)) to encapsulate the result - a list of Concepts, their distance from the original Concept plus an optional Relation (used for *getAllConceptsByPath* when no Relation is given in the request). The result of these two methods is an array of *ConceptRelatives*, one array per each distance (1..n).

2.6. SKOS API Documentation -

The SKOS API is documented in [⌊ Javadoc](#) from the original Java sources. The [⌊ SKOSThesaurus](#) class contains the methods of the web service with the other classes providing the types, as discussed in the previous sections. These are then processed by Java2WSDL to produce the [⌊ http://www.w3.org/2001/sw/Europe/reports/thes/api/wsd/Service.wsd](#) file which is WSDL and not user documentation.

3. SKOS Thesaurus Demonstrator

The demonstrator is called DREFT: Demo of RDF Thesaurus and is primarily the server providing the SKOS API but also includes an example client that uses the API.

3.1. DREFT Thesaurus Web Service Server -

The demonstration server that implements the SKOS API was written in Java using the the open source [Tomcat](#) Java servlet container applications server, [Apache Axis](#) (Java) for web service integration including SOAP de/serialization, and the [Sesame](#) RDF toolkit providing storage, indexing and query with a MySQL-backed triple store.

The server was loaded with SKOS data for two thesauri - GEMET [\[GEMET-SKOS\]](#) supplied by EIONET along with a version of the UK Government Category List (GCL) [\[GCL\]](#) both originally created by Alistair Miles for the SWAD Europe Thesaurus activity.

The demonstration is accessed at:

- <http://thes.ilrt.bris.ac.uk/SKOSThesaurusService> as a SOAP web service intended for machine use described by the WSDL at <http://www.w3.org/2001/sw/Europe/reports/thes/api/wsd/Service.wsd>
- A [web-based browse of the underlying RDF graph](#) using the standard Sesame web interface. This allows queries to be made with SerQL, RDQL or RQL.

The demonstration web service presently does not implement methods for searching for concepts by regex and running an RDF query (although one can do this via the native Sesame interface). It also does not use a free text indexer so the keywords search is very slow as it looks at all triples for matches. The "ByPath" methods are limited to a maximum of 3 steps in order to limit the server load.

The server package may be downloaded from the [DREFT download area](#). A 'README' file is included with the server download and includes an explanation of how to install and configure this release of the software, as well as how to *further develop* the software in future iterations, following its extensible design pattern. Also included is a Java test client with some example data.

3.2. DREFT Thesaurus Web Service Client -

A demonstration client calling the web service has been written in Python. This shows that the interface to the service is truly separate from the implementation, mediated by the web service description provided in the WSDL. Python also has some support for generating methods and types from the WSDL, in the same fashion that is typically expected in more typical web-service implementation languages such as Java and the *.Net* languages.

The client demonstration is a small program that consists of several wrapper classes to invoke the web service calls, along with formatting of the results in HTML and run as a CGI script.

The client demonstration is at:

- <http://thes.ilrt.bris.ac.uk/demos/get>

and includes links to the python sources.

The client package may be downloaded from the [DREFT download area](#) along with the server and example data.

3.3. Thesaurus Example data -

The GEMET [\[GEMET-SKOS\]](#) data supplied by EIONET was used for the prototype work, with a slight update to provide `skos:inScheme` triples so that the

inThesaurus methods worked. The UK Government Category List (GCL) [\[GCL\]](#) data was also loaded. Both of these were created by Alistair Miles for the SWAD Europe Thesaurus activity.

The amended data may be downloaded from the [DREFT download area](#) along with the client and server.

4. Issues

The SKOS API is an initial attempt at a web service for the SKOS-Core. There are still open issues, as one API cannot provide all that is needed for all use cases, or even all the use cases originally considered.

- Maybe add 7 more methods like GetXYZByURI and GetXYZbyExternalID so that the client doesn't have to use a Concept at all as a parameter.
- In our implementation, there is a tension between using the SOAP encoding model or document literal with RDF/XML (for example) to represent concepts and relations. It might be better to use both as appropriate.
- We tried to use `xsd:anyURI` instead of a separate URI class but the Java2WSDL tools we are using made this hard. It may be possible to post-process to do this if this is an advantage.
- Versioning of the API. Should we add a `getVersion()` method?
- We would like to provide access to RDF query support where available (like Joseki) since this is over an RDF model. This may link to the W3C [DAWG](#) work.
- We really want a REST API especially as this is a non-side effecting API, so HTTP GET operations would be safe.
- What fields of concepts should the regex and keyword search methods match?
- Modeling issues about return values from such things like returning concept relatives - what kind of concept and relation should be returned. We return two forms of result presently - lists of concepts or lists of `ConceptRelatives`.
- Support for SKOS mapping.
- Better searching for text and RDF literals along with or across languages.
- Consideration of providing direct RDF query language support such as being standardised by the W3C RDF DAWG. Since Sesame already provides several query languages, this is more an API issue.
- Currently an application-specific issue (related to our use of Sesame) is whether to use the SKOS rule file we created to support transitivity for SKOS relations, separately from the alternative RDF-S-based rules file we currently use for our implementation. In other words, whether to use two separate databases (conceptually - from Sesame's point of view) each to cater for the different levels of inferencing required by certain API methods. The former supports, for example, querying for `TopConcept Concepts`, but it is a problem for API methods that seek to return `ConceptRelatives` (the reason being that those methods are implemented by iterative method calls for 'immediately' (1-distance-away) related Concepts as opposed to requiring inferencing over SKOS relations).

References

[SKOS GUIDE]

SKOS-Core 1.0 Guide. Miles, A.J., Rogers, R., Beckett, D. SWAD-Europe Thesaurus Activity.

<http://www.w3.org/2001/sw/Europe/reports/thes/1.0/guide/>

[WSIBP]

WS-I Basic Profile Version 1.0, Final Material 2004/04/16. K. Ballinger, D. Ehnebuske, M. Gudgin, M. Nottingham and Prasad Yendluri. This version is

<http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html>. The [latest](#)

version is <http://www.ws-i.org/Profiles/BasicProfile-1.0.html>.

[GEMET-SKOS]

└ GEMET data in SKOS, A. Miles and S. Roug. European Environment Information and Observation Network (EIONET), 2004-06-14.

[GCL]

UK └ GCL (Government Category List) V2.1, UK GovTalk, 2004-07.