DCPAPERS

Towards Terminology Services: experiences with a pilot web service thesaurus browser

Douglas Tudhope
Hypermedia Research Unit
University of Glamorgan
Wales, UK
Tel: +1443 482271
Fax+1443 482715
dstudhope@glam.ac.uk

Ceri Binding
Hypermedia Research Unit
University of Glamorgan
Wales, UK
Tel: +1443 482271
Fax+1443 482715
cbinding@glam.ac.uk

Abstract:

The lack of standardised access and interchange formats impedes wider use of terminology resources on the Web. General programmatic access requires commonly agreed protocols. The paper discusses initial experiments with a pilot web service demonstrator for the SKOS API protocol. This 'rich client' browser displayed details for concepts from the GEneral Multilingual Environmental Thesaurus (GEMET). The user interface is described and the importance of caching for fast response emphasized. Limitations of the current system are discussed and directions for future work outlined. The API appears to support client applications of this type but further refinement of the API is required.

Keywords:

Knowledge Organization System, KOS, Thesaurus, Web Service, API, Browser

1. Introduction

Dublin Core recommends controlled terminology for the Subject of a resource. Knowledge Organization Systems (KOS), such as classifications, gazetteers, taxonomies and thesauri, provide controlled vocabularies which organize and structure concepts for indexing, classifying, browsing and search. Thesauri are one of the most commonly used KOS. A thesaurus employs a set of standard semantic relationships (ISO 2788, ISO 5964) and major thesauri will have a large entry vocabulary of terms considered equivalent for retrieval purposes. Many KOS have been made available for web based access. However, they are often not fully integrated into indexing and search systems and the full potential for networked and programmatic access remains untapped.

The lack of standardised access and interchange formats impedes wider use of KOS resources. For example, we developed a web demonstrator (3) for the FACET project (2) which explored thesaurus-based query expansion with the Getty Art and Architecture Thesaurus. The browser-based interactive interface permits dynamic control of query term expansion (Figure 1). However, being based on a custom thesaurus representation and API, the techniques cannot be applied directly to thesauri in other formats on the Web.

General programmatic access requires commonly agreed protocols, for example building on Web and Grid services. The development of common KOS representation formats and service protocols are closely linked. Linda Hill and colleagues have argued for 'a general KOS service protocol from which protocols for specific types of KOS can be derived' (5). Thus, in future a combination of thesaurus and query protocols might permit a thesaurus to be used with a choice of search tools on various kinds of database.

Service oriented architectures bring an opportunity of moving towards a clearer separation of interface components from the underlying data sources. The FACET web demonstrator was implemented via Active Server Pages (ASP) with server-side scripting and compiled server-side components for database access, and cascading style sheets for presentation. This experience led to the conclusion that basing distributed protocol services on the atomic elements of thesaurus data structures and relationships is not necessarily the best approach; client operations that require multiple client-server calls would carry too much overhead (1). This would limit the interfaces that could be offered by applications following such a protocol. Advanced interactive interfaces require

https://doi.org/10.23106/dcmi.952108312

protocols which group primitive thesaurus data elements (via their relationships) into composites, to achieve reasonable response.

1.1 SKOS API and Schema

The Simple Knowledge Organisation System (SKOS) API is a recent development, which addresses some of these issues. It defines a core set of methods for programmatically accessing and querying a thesaurus based on the SWAD-Europe project's SKOS-Core RDF schema (6, 8). The API is an interface designed to provide programmatic access to thesauri and other simple knowledge organisation systems (SKOS) via the web, provided they are represented according to the SKOS-Core schema. The API consists of a number of function calls to access data from a given thesaurus (9). While intended as web service calls, the API itself remains independent of such concrete implementation details.

The SKOS API builds on previous protocols (see 1 for a review of CERES, Zthes and ADL). Briefly, one set of SKOS calls returns a concept(s) with its details via an ID, a preferred label, or matching a keyword or regular expression. One call returns a list of supported semantic relations for the given thesaurus. Another set of calls returns concepts connected by a specified relation or all immediately connected concepts. Importantly, it is possible to get a set of concepts connected by a relation up to a given path length.

2. Pilot SKOS API Browser

We developed a pilot PC based (.NET) web service client application as an initial experiment with the SKOS API, a 'rich client' browser displaying details for thesaurus concepts. It made SKOS API calls to the SWAD-Europe DREFT web services server at the (Bristol University) Institute of Learning and Research Technology (ILRT). DREFT was a temporary demonstrator for aspects of the SKOS API and the software is freely available for download (9). We utilised the ILRT DREFT Server to experiment with a remote server.

The browser acted on GEMET (GEneral Multilingual Environmental Thesaurus), held on the DREFT server in SKOS format - see http://www.eionet.eu.int/gemet.

Existing SKOS API calls would have returned thesaurus relationship data. This would have allowed an interface similar to the FACET web demonstrator, where hierarchical and associative relationships are visualized and can be navigated. However, due to limitations imposed by local requirements on the DREFT server configuration at the time, these API calls were disabled. SKOS API calls that involved string matching of concept terms were also not available at the time, so matching of user search statements with controlled terminology was not possible.

Therefore the browser only utilised a small subset (two) of the possible SKOS API calls: *getConcept* and *getAllConceptRelatives*. These calls do not return relationship information so the browser could only display immediately semantically related concepts in a linear list of concepts with *some* direct relationship to the displayed concept.

2.1 User Interface

The user interface (Figure 2) comprised a single screen. “Back” and “Next” buttons provided functionality similar to web browser history buttons – navigate back and forth through a sequence of previously viewed concepts. This was relatively fast since previously retrieved concepts were cached locally during a session - further server calls were not necessary.

Figure 2: SKOS API browsing client

Directly below these buttons, the “Concept Reference” section held a drop-down list of concept identifiers for all previously viewed concepts and a “Go” button, which displayed details for the concept represented by the selected identifier. Some initial GEMET identifiers were provided at start-up, or an identifier could be typed directly into the box if known. The identifiers formed convenient “jumping in” points to initiate a browsing session, in the absence of any comprehensive concept search facilities. As

https://doi.org/10.23106/dcmi.952108312

each new concept was viewed, its identifier was added to the drop down list for future re-selection.

It is important to note that although concept identifiers were structured as URI's they were not necessarily real, accessible web locations – the browser application used web service API calls to retrieve data using the selected concept identifier as a unique reference to the resource required. The "Concept Details" section consisted of 3 separate boxes:

1. The first box displayed the preferred term (in bold) for the currently selected concept. Any non-preferred terms for the currently selected concept were also displayed in square brackets.
2. The middle box displayed any scope notes for the currently selected concept.
3. The lower box listed concepts with *some* direct relationship to the currently displayed concept. These concepts were displayed in ascending alphabetical order, together with a count. Clicking any item in this list retrieved and displayed the details for that concept, replacing the previously displayed concept – effectively allowing the user to "browse" through the concept space. Concepts not previously viewed were shown in blue type; concepts previously visited were shown in purple type.

The status bar at the bottom of the form (blank in Figure 2) gave general feedback on the status of current operations – in particular indicating when a call was being made to the server. In operation we found that although most of the time server response was very good, occasionally a call took some time to return data (10 seconds or more) and the application waited for a response in order to update the display.

2.2 Caching

Based on early experimentation it was clear that caching (local storage) of concepts would be beneficial to prevent unnecessary repeated server calls. The current design of the SKOS API dictates that two separate server calls are necessary for the display of a concept and its directly related concepts. Bearing in mind the relatively static nature of the data being viewed (i.e. the underlying thesaurus data is unlikely to be changed on the server during a single user session); local caching of concepts for the lifetime of the user session was viewed as a sensible strategy. The implementation of concept caching made a significant difference to the apparent speed of operation, enhancing the overall user experience.

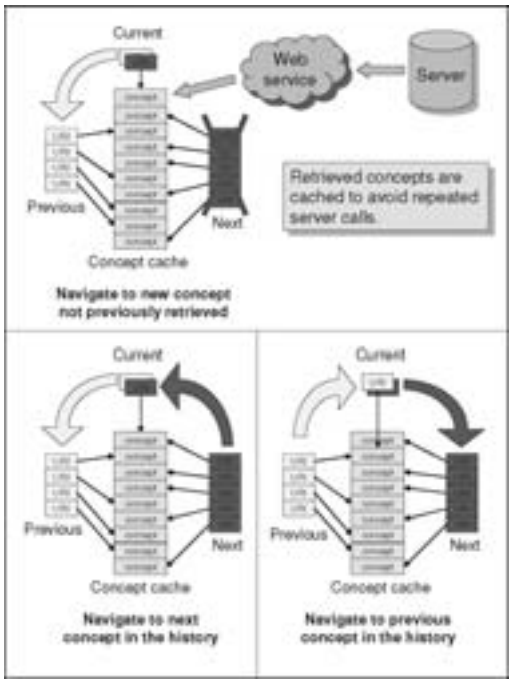

Figure 2: SKOS API browsing client

Concepts are always retrieved from the local cache for display (Figure 3). Any request for a concept first consults the cache. If the concept already exists it is displayed, otherwise it is retrieved from the server and added to the cache, then displayed. Concepts in the 'previous' and 'next' browsing history are guaranteed to already exist in the cache.

3. Discussion and recommendations

We intend to continue this line of research and explore web service clients with richer functionality that utilize more API calls. Future issues for Thesaurus and KOS protocols include possible provision of more complex services, such as semantic expansion, more advanced natural language functionality, cross-mapping provision, and data-dependent filters such as the number of postings associated with a concept.

When considering large-scale, augmented KOS or ontologies that are represented as a semantic network (as an RDF triple store) then general semantic query languages such as SPARQL, RDQL, SeRQL and RQL are appropriate for applications requiring logic-based reasoning. On the other hand, the SKOS API builds on work in the NKOS (7) and DL communities on use cases in KOS-based services, with a strong interactive component. Given foreseeable bandwidth limitations, KOS-specific protocols continue to be needed for terminology services. Future API designs should evolve to support the integration of KOS into rich but responsive mapping and query terminology services,

https://doi.org/10.23106/dcmi.952108312

with flexible expansion and visualisation capabilities.

With regard to evolution of the API, we recommend that protocols such as the SKOS API return relationship information with all calls. Those calls that return a list of concepts related by a specified relationship (eg broader) should return a structured list, identifying the level of expansion from the initial concept, rather than an undifferentiated list as at present. A single call should be available for the display of a concept and its directly related concepts.

4. Conclusions

As initial experimentation with the SKOS API, we developed a pilot PC based (.NET) web service client demonstrator application. This was tested on a remote server, which used a different technology platform. Supported by concept caching, it generally achieved a fast enough response for reasonable interaction and suggests that the SKOS API can support client applications of this type. However further refinement of the API is required.

References

1. C. Binding and D. Tudhope D. KOS at your Service: Programmatic Access to Knowledge Organisation Systems. In: Journal of Digital Information, 4(4), 2004. http://jodi.ecs.soton.ac.uk/Articles/v04/i04/Binding/
2. FACET Project, University of Glamorgan. http://www.comp.glam.ac.uk/~facet/facetproject.html
3. FACET Web Demonstrator. http://www.comp.glam.ac.uk/~FACET/webdemo/
4. Glamorgan web service pilot browser. http://www.comp.glam.ac.uk/~FACET/services/
5. L. Hill, O. Buchel, G. Janée and M. Zeng. Integration of Knowledge Organization Systems into Digital Library Architectures. Proc. 13th ASIS&T SIG/CR Workshop, *Reconceptualizing Classification Research*. 2002.
6. A. Miles, B. Matthew, N. Roger, D. Beckett. SKOS: Standards and Best Practises for Using Knowledge Organisation Systems on the Semantic Web. NKOS Workshop, Bath. 2004. http://www2.db.dk/nkos-workshop/pp%20presentationer/SKOS_NKOS2004.pdf
7. NKOS Network. Networked Knowledge Organization Systems/Services.
   - http://nkos.slis.kent.edu/
8. SKOS http://www.w3.org/2004/02/skos/
9. SKOS API. http://www.w3.org/2001/sw/Europe/reports/thes/skosapi.html

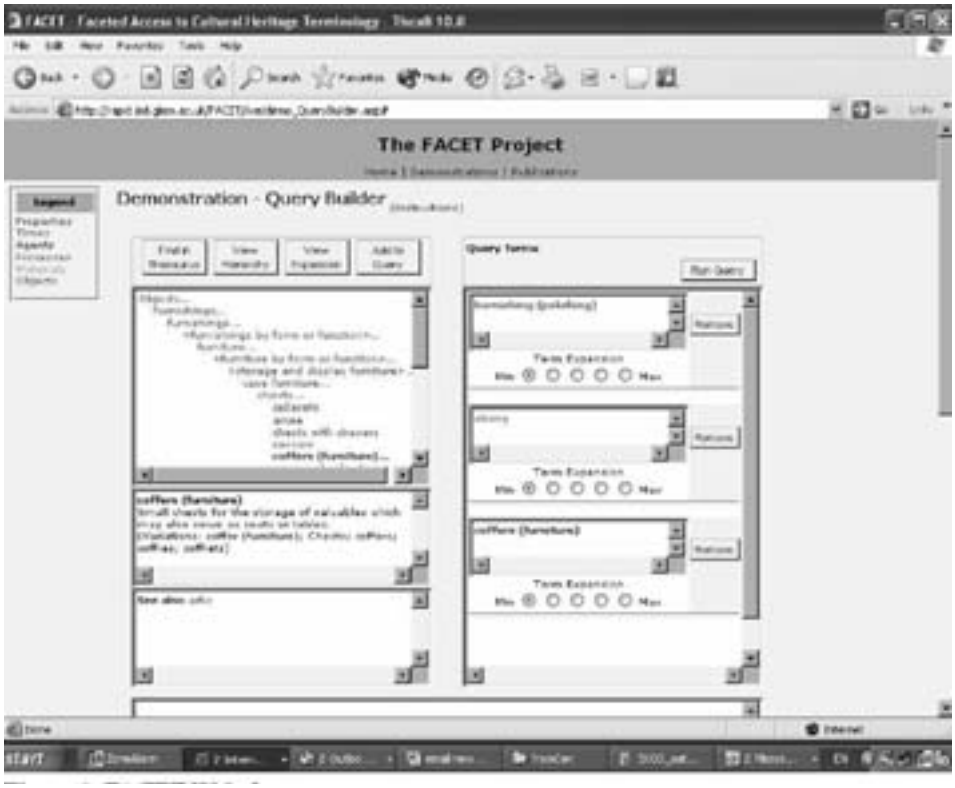

Figure 1: FACET Web demonstrator

https://doi.org/10.23106/dcmi.952108312