A Flexible API and Editor for SKOS

Simon Jupp, Sean Bechhofer, and Robert Stevens

School of Computer Science, University of Manchester, UK
`first.last@manchester.ac.uk`

Abstract. We present a programmatic interface (SKOS API) and editor for working with the Simple Knowledge Organisation System SKOS. The SKOS API has been designed to work with SKOS models at a high level of abstraction to aid developers of applications that use SKOS. We describe a SKOS editor (SKOSEd) that is built on the Protege 4 framework using the OWL and SKOS API. As well as exploring the benefits of the principled extensibility afforded by this approach, we also explore the limitations placed upon SKOS by restricting SKOSEd to OWL-DL.

1 Introduction

We present an editor for The Simple Knowledge Organisation System (SKOS) called SKOSEd¹ and an associated SKOS API². SKOSEd uses the SKOS ontology as a schema to drive the API, editor and the development of SKOS vocabularies. In doing so, SKOSEd takes advantage of existing OWL 2 DL infra-structure and gains the benefits of a *principled* means of extending SKOS vocabularies through committing to an OWL 2 DL version of SKOS.

SKOS is a developing specification for representing and publishing classification schemes, thesauri, taxonomies and subject heading systems on the web. It is currently under development as part of the W3C Semantic Web Deployment Group (SWDWG)³ and Editor's Working Drafts of the SKOS Reference⁴ and SKOS Primer⁵ are available. In order to facilitate adoption and use of SKOS there is now a need for tools that can support its use in the form of APIs, editors, browsers and validators.

There has been a long tradition in the library and information sciences in using thesaurus-like artefacts to index and manage large volumes of data, such as inventories or cataloging systems [10], [12], [9]. For the purposes of this paper we will refer to these artefacts as Knowledge Organisation Systems (KOS). Applications of KOS include indexing; search and information retrieval; text processing (e.g. text mining); and document collection navigation.

¹ <http://code.google.com/p/skoseditor/>

² <https://sourceforge.net/projects/skosapi/>

³ <http://www.w3.org/2006/07/SWD/>

⁴ <http://www.w3.org/TR/skos-reference/>

⁵ <http://www.w3.org/TR/skos-primer>

These KOS can span multiple domains and vary considerably, as they are often designed for a particular application or institution. Despite this, they often share many features in common, especially in their content type and structure. As we move towards a Semantic Web, providing structure to content on the web, a lot can be gained from exploiting existing KOS artefacts and technology. Technologies such as RDF and OWL [6] are seen as key elements for building a Semantic Web. The SKOS model is built in accordance with these technologies and has a serialisation to the Resource Description Framework (RDF). In general, KOS differ significantly from formal ontologies, as represented using OWL, as they do not contain detailed intensional descriptions of concepts [3], and often contain relationship between concepts where the semantics are weakly defined or ambiguous [4].

The SKOS model can be used to structure and represent any knowledge artefact that contains statements about concepts and the relationships between them. The shared features of these KOS are primarily in the form of a lexical resource along with some semantic relationships between each resource. The semantic relationships between resources are typified by synonym, hypernym, hyponym, antonym, broader, narrower, and related.

SKOS provides a data model that can be used to express these kinds of relationships between resources and is designed to be extensible and modular. Central to SKOS is the core vocabulary deemed sufficient to represent most of the common features found in concept schemes. A *concept* can be considered any unit of cognitive thought. *Lexical labels* allow the association of lexical forms (preferred labels, alternative labels and so on) with each concept. *Semantic relations* capture relationships between concepts including hierarchical broader-narrower relationships and general associative relationships.

A goal of SKOS is to provide flexibility—demands and expectations for what a KOS should provide vary across communities and different aspects of its requirements will be reflected in how different tools support its use. There are a community of users who wish to use SKOS to represent a complete KOS, which could then in turn be used as a reference system for providing annotations. SKOS also provides vocabulary for metadata on the web—for example, its use as a meta-data vocabulary for annotating OWL ontologies is one area where SKOS is seen as of potential use [8].

Standardised representations are key to interoperation, but additional support is needed in order to build applications that make use of these languages. In particular, language APIs, frameworks or toolkits facilitate the construction of applications and services, allowing developers to work at a higher level of abstraction, and providing some insulation from issues relating to serialization or parsing of data structures. For example, frameworks like Jena⁶ or the OWL API⁷ [2] provide code libraries for working with RDF⁸ or OWL⁹.

⁶ <http://jena.sourceforge.net>

⁷ <http://owlapi.sourceforge.net>

⁸ <http://www.w3.org/RDF>

⁹ <http://www.w3.org/2004/OWL/>

SKOS is defined as an RDF vocabulary, and vanilla RDF infrastructure could be used to support SKOS in applications. As with earlier work on OWL [2], we, however, argue that appropriate abstractions facilitate application development. In this paper we describe work towards such abstraction in an API for SKOS.

The flexibility of SKOS is, in part, given through its specification and description in OWL Full (See section 2). This then allows further specialisation of the vocabulary in a *principled* manner—for example through the introduction of sub-properties of the SKOS semantic relations. Through the use of existing OWL 2 DL infrastructure, we are able to provide a flexible collection of classes for the manipulation of SKOS models. As an illustration of the use of the API, we present a prototype SKOS editor – SKOSed – and discuss plans for a larger suite of tools providing SKOS validation and support for conversion between SKOS and OWL. Our approach brings a number of benefits:

- Support for extension of the basic SKOS vocabulary through the definition of subproperties;
- Basic reasoning about inverse relationships;
- Characteristics of properties can be asserted (for example functionality or transitivity), with corresponding effects on the results of queries;
- We have access to additional expressivity. For example the use of property chains in order to capture additional semantics, such as the interaction between locative and partitive properties.

In all the above cases, the use of the underlying OWL 2 DL infrastructure provides easy access to reasoning services that support the use and extension of the vocabulary.

2 SKOS and OWL

Standardised representations such as RDF and OWL are key to the Semantic Web vision. RDF provides a common data model and syntax for metadata representation on the web. RDFS and OWL provide schema languages for modeling data on the web. OWL can be used for modeling domain knowledge and has a formal semantics for building expressive and powerful knowledge artefacts. The style of knowledge artefact found in most KOS often lack a precise formal semantics that can make direct conversion into an OWL TBox inappropriate [13]. KOS are really just facts asserted about things (concepts) – an OWL ABox (a collection of axioms about individuals) is thus more appropriate. We can, however, use an OWL TBox as a meta-vocabulary and a data model for KOS and use OWL inference to aid construction, maintenance and use.

A combination of RDF and OWL are used to define the SKOS data model. The data model is itself an OWL ontology, any thesaurus represented in SKOS is then an instantiation of this ontology. The SKOS ontology provides a minimal vocabulary for representing a KOS, this vocabulary can be extended with specialisations of the SKOS constructs. A useful type of extension to SKOS may be extending the basic *Semantic Relationships* – **broader**, **narrower**, and **related**

– that are asserted between **Concepts**. The **broader** and **narrower** relationships provide some hierarchical structure to a KOS, however, it can be desirable to express different kinds of broader/narrower relationships, for example, extensions to **broader** such as **broaderGeneric** and **broaderPartitive**, to specify a class subsumption and partitive relationship between **Concepts** respectively. The ability to extend SKOS gives KOS authors the flexibility to more accurately represent their own knowledge artefact, whilst keeping it defined within the SKOS model for interoperation with other SKOS aware applications on the web. The ability to extend the SKOS data model in terms of OWL is a key motivator for using existing OWL infrastructure as the basis for the SKOS API and SKOS^{Ed}.

The SKOS data model is described using OWL Full, the most expressive OWL species. OWL Full has few constraints on the use of the vocabulary, providing a useful and flexible language for describing data models. The existing SKOS data model is designed with flexibility in mind, and uses elements of OWL Full where necessary. While this approach has benefits for SKOS as an expressive language for describing a KOS, it does have implications for applications and tools that wish to support SKOS. This is mainly down to the fact that SKOS is inherently coupled to the RDF syntax, it does not yet have a complete data model that is abstract from this syntax. Any dedicated SKOS editor would need to provide support for not only SKOS, but also the OWL Full and RDF infrastructure. Current support for OWL Full editors is limited and the interfaces to these application are not necessarily familiar to users of conventional KOS software. By defining a restricted, yet valid, SKOS model that is abstract from the RDF syntax we are able to provide support in the form of an API and editor for SKOS that is more akin to standard KOS editing software.

We have taken inspiration from the ontology community where the use of a restricted sub language of OWL Full, called OWL 2 DL, has been used to define an abstract data model for OWL ontologies that also benefits from being based on a Description Logic (DL). The OWL 2 DL sublanguage describes a model that can be abstracted from any concrete syntax, such as RDF, and has been exploited in tools for working with ontologies, such as the OWL API[2,5] and Protégé 4¹⁰. The majority of constructs defined in the SKOS model can be expressed using OWL 2 DL, the work presented here aims to exploit tools currently available for working with OWL 2 DL ontologies, and build a similar infrastructure for working with SKOS, or even combinations of both OWL and SKOS.

One of the main defining features of the OWL 2 DL sub language is the ability to use DL reasoners to compute entailments and check consistency [11]. Although the expressivity of SKOS is only a small fraction of that which is capable of use by current reasoners, we can take advantage of tools that exists to infer any entailments that can be made using the SKOS semantics. The kinds of entailment that can be computed within SKOS are relatively simple in comparison to what is capable with OWL 2 DL. They are, however, useful and should still be

¹⁰ <http://protege.stanford.edu>

Asserted facts

```

a rdf:type skos:Concept
b rdf:type skos:Concept
ex:has-part rdf:type owl:ObjectProperty
ex:part-of rdf:type owl:ObjectProperty
ex:has-part rdfs:subPropertyOf skos:narrower
ex:has-part owl:inverseOf ex:part-of
a ex:has-part b

```

Inferred facts

```

ex:part-of rdfs:subPropertyOf skos:broader
b ex:part-of a

```

Fig. 1. Example of inferences that follow from the SKOS semantics

supported by the API and SKOSEd. Consider the example shown in figure 1, **Concept a** is asserted to be related to **Concept b** via a **has-part** relationship. The **part-of** relationship is a sub-property of the SKOS **narrower** property. We can use the fact that SKOS **broader** and **narrower** properties are defined as **inverse** properties in SKOS, to query for all the concepts that are related to **Concept b** via **part-of**. Although this is not asserted in our vocabulary, we can use the semantics of SKOS to infer that **Concept b** is related to **Concept a** via **part-of**.

There is great potential for combining OWL and SKOS in bridging the gap between domain modeling in OWL and language models built in SKOS. Building a knowledge base that contains an expressive OWL ontology combined with a model of the language in SKOS is potentially powerful and accommodates various axes for query and navigation of the information space, keeping SKOS in OWL 2 DL is likely to make this more feasible.

3 API Design

Many KOS authors interested in publishing data on the web may prefer not to be exposed to the underlying technical details such as RDF and OWL. We suggest that a SKOS data model that provides an appropriate abstraction from the technical issues of concrete representations (such as RDF) is of benefit. This approach proved successful in designing an API for OWL [2] and subsequent tools such as an OWL Validator [1] and editor, Protégé¹¹. Such tools are of benefit to the growing community of OWL users, and it is hoped that building a suite of tools for working with SKOS will have similar benefits.

The first step in designing the API and building a subsequent editor on top of that API is to define an abstract model for SKOS.

The key issues addressed in this paper are about providing access to the underlying data model to support extensions, and to have built in support for

¹¹ <http://protege.stanford.edu/>

interpreting the SKOS semantics, and how this requirement can be achieved using OWL. Many tools exist for working with RDF, including APIs¹² and graphical editors¹³, however, these often operate at too low a level, exposing the underlying RDF structures which may be unfamiliar or unintuitive to KOS developers. Tools exist for building KOS, yet few have direct support for SKOS or provide any programmatic interface to the model. With these considerations in mind, our approach was to exploit current OWL tools and build an API and editor on top of these.

Protégé is the most widely used open source ontology editor available to the OWL community. The latest version of Protégé, Protégé 4, is built on top of the OWL API [5]. The OWL API is a new implementation of the original Wonder Web API [2] and has full support for the emerging OWL 2 DL standard¹⁴. The API provides interfaces to an abstract representation of the OWL constructs for working with OWL ontologies. It comes with an in-memory implementation and supports a wide range of parsing and rendering services in addition to access to DL reasoners. Given our commitment to providing access to the underlying OWL model from within the API, and our desire to use some kind of inference engine to handle SKOS entailments, we chose to build an extension of the OWL API to support SKOS. In addition to the advantages the OWL API provides in its own right, its relationship to the Protégé interface provides us with an existing platform upon which to build a SKOS editor.

The SKOS schema is OWL Full – primarily due to the existence of subproperty axioms concerning annotation properties¹⁵. As a result of this, our API and editor will not support arbitrary SKOS/RDF vocabularies. All vocabularies produced using the tool will be valid SKOS, but there may be cases where arbitrary vocabularies produced outside of the tools may not be processable. We are taking a pragmatic approach here, and hope that the advantages we gain through the reuse of the OWL API and Protégé framework outweigh requirements to support arbitrary SKOS/RDF vocabularies.

It is likely that the OWL 2 DL specifications will address many of the issues that result in SKOS being OWL Full (for example through the support for punning and rich annotations). As the OWL API is committed to supporting OWL 2 DL, we believe that this will then enable us to support and handle the vast majority of SKOS vocabularies.

Our choice to use the OWL API as the basis for the SKOS API is driven by our requirements to have access to the OWL data model that will support the ability to move between pure SKOS and a combination of SKOS and OWL. The SKOS API provides a set of interfaces for working with SKOS along with an implementation that uses the OWL API. Separation of the interface from

¹² <http://jena.sourceforge.net/>

¹³ http://www.altova.com/products/semanticworks/semanticweb_rdf_owl_editor.html

¹⁴ <http://www.w3.org/2007/OWL>

¹⁵ Based on the latest Editor's Draft revision of the SKOS Reference, <http://www.w3.org/2006/07/SWD/SKOS/reference/20081001/>, revision 1.45 as of 15/12/200

Fig. 2. Abstraction of the SKOS data model showing the base object we have in the SKOS API, these have further specialisations (see API documentation for full details)

implementation leaves the API open for others to provide their own backend implementation. Figure 2 is an object diagram showing the basic SKOS data model represented in our SKOS interface.

The interface provides a set of common objects for building and querying a SKOS model. Most of these are convenience methods for working with the types of object found in SKOS. These interfaces alone could be implemented with any of the popular APIs for working with RDF; we chose to use the OWL API and have extended only the interfaces needed to work with SKOS. The implementation is essentially a wrapper around the OWL API, along with a set of convenience methods for working with SKOS. Access to the underlying OWL API objects that are being manipulated is always available via the *SKOS to OWL Converter* utility class, which may be desirable when exploiting the ability to extend the SKOS model. We also reuse the practice adopted by the OWL API to separate asserted and inferred models. A DL reasoner can be used to provide the inferred SKOS model for more sophisticated querying.

4 API Functionality

Much of the key functionality of the SKOS API is provided *via* the use of the OWL API.

Parsing and Rendering. We take advantage of the OWL API serialisation and parsing mechanisms which support handling of multiple concrete syntaxes for SKOS. For the most part RDF/XML will be the default and the most common.

Change control. Manipulation of a SKOS via the addition and removal of objects and assertions. This includes a mechanism for tracking changes that are made to a SKOS.

Extension to the model. The OWL API provides interfaces for manipulating the underlying OWL ontology and SKOS model, our implementation provides support for using the OWL API objects with the SKOS API.

Inference. We can exploit the power of DL reasoners to build an inferred model of a SKOS data set. The API provides support for working solely with an asserted model, or querying an inferred model.

Overall control and manipulation of SKOS is handled by a *SKOSManager* class. The *SKOSManager* is responsible for loading, creating and saving SKOS data sets. A *SKOSManager* can contain multiple SKOS data sets. The SKOS data set contains all the assertion about a particular KOS, this may include multiple concept schemes and sets of concepts that have some relationships asserted on them. We propose a set of SKOS objects in the API to represent those types of object contained in the data set.

We have strived to keep the API simple and ensure its usage is consistent. The SKOS API is structured as a hierarchical set of SKOS Objects. Four main types of objects are used in combination to build the SKOS model. These are: (i) *SKOSEntity*, (ii) *SKOSRelationship*, (iii) *SKOSAssertion* and (iv) *SKOSLiteral*. These objects are further extended to some more specific types. All objects are accessed *via* an object factory – *SKOSFactory*. Assertions are built using SKOS objects and added to data sets using the SKOS Manager.

SKOSEntity. A SKOS entity represents the major types of objects found in a SKOS, this includes *Concept*, *ConceptScheme*, *Collection* and *LabelRelation*. As the SKOS reference is finalised, more entities may be added. We also plan to provide support for SKOS extensions proposed by the SWDWG. These Classes have convenience methods for accessing the kinds of SKOS data asserted on these entities, for example, a *SKOSConcept* has methods for accessing related concepts and documentation properties such as preferred label and definitions.

SKOSProperty. The *SKOSProperty* class is extended to represent the major types of properties found in SKOS. For example, *SKOBroaderProperty*, essentially an OWL Object Property and the SKOS label properties that are OWL Data Properties. We also have the SKOS document properties that can be used as either OWL object, data or annotation properties. Despite the OWL full implication of this use of property punning, we are able to handle this within the OWL API.

SKOSLiteral. The *SKOSLiteral* class represents our RDF literals, these can be typed, such as String, or untyped, with the untyped values having support for XML language tags. These are usually used as the filler of a relationship along a data or annotation property.

SKOSAssertion. *SKOSAssertion* is used to represent relationships between SKOS Objects. These are further divided into SKOS object, SKOS data and SKOS annotation property assertion. An example of a SKOS object assertion

might be to make a broader relationship between two concepts. Each SKOS change is enacted with respect to a particular SKOS data set *via* the manager. The use of explicit change objects supports the management of changes being applied to a particular dataset (as discussed in [2]). This is particularly useful in the context of editing, as it facilitates the support of undo and redo actions.

SKOSReasoner. *SKOSReasoner* is an interface to query an inferred representation of the SKOS dataset. This class can take any DL reasoner that implements the *OWLReasoner* interface from the OWL API. There are currently three publicly available reasoners that implement the *OWLReasoner* class that can be used to classify SKOS dataset; Pellet¹⁶, Fact++¹⁷ and HermiT¹⁸. Whilst all three reasoners are more than capable of processing the simple entailments that can be inferred using SKOS semantics, all three systems vary considerably in terms of memory consumption and speed.

5 SKOS API Performance

In order to demonstrate the capabilities of the SKOS API we ran some simple tests with a set of publicly available thesauri. Scalability is still a big issue with Semantic Web technologies and as thesauri have a tendency to be rather large, it is important that SKOS tools are scalable. We chose five thesauri that have been converted into a SKOS representation, we chose these based on their public availability, size and the fact that they use more than SKOS's basic expressivity. We measured the time it takes for the SKOS API to read the RDF files into memory and the amount of memory consumed. These test were performed on an Apple Mac Pro with two 2.2 GHz Dual-Core Intel processor, the Java virtual machine was limited to 2GB of addressable memory.

- GTT¹⁹ thesaurus is used to index and retrieve books from the Dutch National Library
- AGROVOC is a multilingual structured thesaurus of all subject fields in Agriculture, Forestry, Fisheries, Food security and related domains.
- The Library of Congress Subject Headings (LCSH) comprise a thesaurus of subject headings, maintained by the United States Library of Congress, for use in bibliographic records.
- Medical Subject Headings (MeSH) is a large controlled vocabulary for the purpose of indexing journal articles and books in the life sciences.
- GEMET is a compilation of several multilingual thesauri that provides a terminology for the environment.

¹⁶ <http://pellet.owldl.com>

¹⁷ <http://owl.man.ac.uk/factplusplus/>

¹⁸ <http://hermit-reasoner.com/>

¹⁹ GTT is a Dutch abbreviation for GOO keyword thesaurus, GOO referring to the so-called Joint Subject Indexing system used by many libraries throughout the Netherlands.

Table 1. SKOS API load times and memory consumption

SKOS data set	Number of SKOS Entities	Number of triples	Time to load (seconds)	Memory consumption (MB)
GEMET	5245	350,357	5	119
MeSH	24,765	615,403	11	215
GTT	65,306	514,619	13	174
AGROVOC	28,440	848,348	24	483
LCSH	266,858	2,441,515	67	845

The SKOS API successfully loaded all our test sets (see Table 1), however, it is worth noting that the parsing and creation of the objects are mostly being handled by the OWL API implementation that sits under the SKOS API. These results show that developing tools for manipulating the largest SKOS dataset using the SKOS API would be no problem on modern desktop machines.

6 SKOSeD

SKOSeD is a plugin for the OWL ontology editor Protégé 4. It aims to provide support for working with SKOS. The decision to use the Protégé environment as a platform for SKOSeD follows on from our decisions and justifications for using the OWL API as a basis for the SKOS API. Protégé is primarily an editor for OWL ontologies and, as mentioned previously, the SKOS model is described in terms of an OWL ontology. Consequently, a standard installation of Protégé is already a capable SKOS editor, however, the interface is not necessarily suitable or intuitive to KOS authors. SKOSeD provides a suite of *Views*, that expose the structure of SKOS, along with support for editing and authoring.

There are a number of existing tools for working with KOS – Willpower Information ²⁰ lists over 40 thesaurus editors alone. Many existing applications use proprietary data exchange formats, lack a programmable interface, and have little or no support for SKOS. One open source editor that does have support for SKOS is ThManager [7]. ThManager is a stand alone application for working with SKOS, and has good support for managing thesaurus meta-data. Two of our requirements for a SKOS editor are currently not met by ThManager: One being the ability to extend the SKOS model and the second being able to exploit the RDFS and OWL semantics. By building SKOSeD on top of the Protégé environment we essentially have this functionality for free.

It is still too early in SKOS development to gather a complete set of user requirements that would satisfy all SKOS communities. The development of SKOSeD is, at least in part, an exercise in gathering these requirements. The SKOS use case and requirements documents ²¹ describes a wide range of requirements from which we can draw ideas about features a SKOS editor should offer.

²⁰ <http://www.willpowerinfo.co.uk/thessoft.htm>

²¹ <http://www.w3.org/TR/skos-ucr/>

These range from simply viewing and editing the core SKOS constructs to more advanced tasks such as manipulating the underlying OWL model and handling inference. SKOSeD is a powerful SKOS editor and offers more functionality than most users new to SKOS will require. The flexible nature of the Protégé architecture should enable development of more lightweight interfaces that essentially hide the more advanced features from new users. We have defined a basic list of functional requirements for SKOSeD that we believe should be the minimum set of features required for any SKOS authoring tool:

- Ability to create, edit and delete SKOS entities such as Concepts, Concept Schemes etc. . .
- Assert SKOS relationships between the SKOS entities, for example `skos:inScheme`, `skos:hasTopConcept`.
- Visualise the broader/narrower hierarchy in the form of a navigable tree.
- Assert and edit labelling properties such as `skos:prefLabel`, along with support for multilingual labeling.
- Support for meta-data such as SKOS documentation properties and other standard vocabularies like Dublin Core.
- Provide alternate renderings in the editor, especially for multilingual thesauri.
- Access to the SKOS data model and support for its extension.
- Query interface that supports SKOS entailments.
- Change tracking, the ability to undo/redo actions and track those changes.
- Ability to parse and render a range of SKOS serialisations, including RDF/XML.

In order to work with SKOS in the Protégé environment we needed a version of the SKOS ontology that was restricted to the OWL 2 DL species. A cut-down OWL 2 DL version of the SKOS schema, has been defined²². This scheme is imported into Protégé *via* the standard Protégé import mechanisms.

Protégé 4 features a set of views, wizards and menu actions that are useful when working with ontologies. Similarly there will be a set of views and actions that are commonly needed for working with thesaurus like artefacts represented in SKOS. The SKOSeD plugin for Protégé 4 is exactly this, a suite of views and tools for working specifically with SKOS. SKOSeD is designed to use the same Protégé API and editor kit that the existing Protégé 4 interface uses, this ensures that all the SKOS views and actions integrate seamlessly with all the default Protégé tools. This flexible plugin framework offered by Protégé 4 is of additional benefit to 3rd party developers who want to extend SKOSeD or add additional tools that would interact with SKOSeD.

One of the core requirements for SKOSeD is a view that displays the hierarchical relationships between Concepts often found in SKOS (See Figure 3). SKOSeD provides a Concept hierarchy view that displays the structure of the hierarchy and allows for its manipulation. For any concept selected in the hierarchy a set of other views are available that display asserted information on the concepts.

²² <http://www.w3.org/2006/07/SWD/SKOS/reference/20081001/skos-dl.rdf>

Fig. 3. A screenshot of SKOS plugin in Protégé, loaded with a SKOS representation of MeSH. The left hand panel shows the Concept hierarchy, the right hand panel shows asserted information about the selected Concept.

Whilst Protégé by default can display any asserted data, object or annotation properties asserted on any individual, SKOSed does this in a more intuitive fashion. Rather than having a long list of assertions, SKOSed provides a separate window for the different types of information asserted on SKOS concepts. Each of these windows has a set of convenience buttons for manipulating SKOS data along with context driven action depending on the type of SKOS assertion being made. This set of views include a view for the semantic relationships, such as broader, narrower and related; a view for adding labeling informations; a view for adding SKOS notes and a default panel for adding any other arbitrary type of relationship that is not part of the SKOS core schema.

In addition to views, SKOSed provides menu items and wizards to simplify some common tasks when creating and editing SKOS. One of these is a wizard to create collections. Collections in SKOS are a meaningful grouping of Concepts. To create a collection in the default Protégé OWL interface the user would have to create a new instance on the Collection class and assert which concepts are members of this collection using the `skos:member` property. SKOSed provides a Collection wizard that allows the user to simply select some Concepts from a list and choose the “Create Collection” action in the menu. Another useful feature is the support for multilingual KOS, Figure 4 shows a portion of AGROVOC that has been rendered along the `skos:prefLabel` property in Chinese. SKOSed allows a user to specify an order of preference for properties and language to render.

Fig. 4. SKOS^{Ed} supports multilingual rendering. The figure shows the asserted Concept hierarchy of AGROVOC with the concepts rendered along the preferred label in Chinese.

SKOS^{Ed} currently follows the Protégé 4 practice of separating the asserted and inferred models. We have chosen to adopt the same practice with the existing SKOS^{Ed} views. For every asserted view available there is a read-only inferred view that shows inferred statements after classification. These inferred views are particularly useful when looking at SKOS datasets that have missing information. For example, we can exploit the inverse nature of the SKOS broader/narrower semantic relationships to display any Concepts that are Narrower than a selected Concept when only broader relationships have been asserted in the SKOS source file. The inferred views are also useful when extensions have been made to the underlying data model. In this way users can create specialisations of the SKOS schema that more accurately reflect their own data model. Whilst the asserted views will not be able to interpret these extensions, the inferred views will allow them to view their data with respect to the extended SKOS data model.

It will take time to gain feedback from users to ascertain how practical this notion of asserted and inferred views is to SKOS^{Ed} users. We take the view that users new to SKOS will not care much for these features initially, the functionality offered by the basic asserted views should suffice for most SKOS development. More advanced SKOS users, who have a better understanding of the underlying OWL data model, will benefit from the inferred views and will also benefit from having access to all the default OWL machinery offered by Protégé 4. The ability to edit and work on inferred models of ontologies, and now SKOS, has long been a problem for tool developers. This is due to the fact that existing reasoners must completely re-classify an ontology after a single axiom change in order to compute any inconsistencies or new entailments. In the past this has not been practical for Ontology developers, especially when the ontologies are large and expressive. Advancements in incremental reasoning are almost at a stage where

asserted views in ontology editors like Protégé 4 may not be necessary, allowing users to work solely on an inferred model. A re-evaluation of the SKOSEd interface will be likely should these tools become available.

7 Discussion

In this paper, we presented an API for working with SKOS, along with a graphical editor SKOSEd, implemented using the API. We discussed the requirements for such tools and our justifications for reusing existing OWL infrastructure to provide support when working with SKOS. We highlighted two key issues that we believe are desirable when working with SKOS; 1/ the ability to extend the SKOS model via an API or editor; and 2/ the exploitation of DL reasoners to compute entailments for richer querying of data represented in SKOS. The first can benefit different communities in allowing the specialisation of SKOS to better represent their data. Additional properties that are extensions of SKOS can provide multiple axes on which to query their knowledge. Access to the underlying data model helps to support this effectively in tools. By reusing an existing API (the OWL API) and ontology editing software (Protégé 4), we were able to rapidly develop early tool support for the emerging SKOS specification.

SKOS is well placed to bridge the gap between the social tagging culture that has already gained popularity on the web, and the kind of rich metadata and structured knowledge provided by ontologies, that is needed to realise a Semantic Web. SKOSEd aims to provide the community with a scalable application for working and managing SKOS data. By adopting the Protégé framework we hope that the existing plugin will be extended with additional plugins provided by the community, to improve the way we work with not only SKOS, but combinations of both the SKOS and OWL languages. The SKOS API provides a flexible set of interfaces for working with SKOS, along with a concrete implementation using the OWL API. The API is independent from the editor and additional tools are planned, including a re-implementation of the interface to provide some persistent storage mechanism, such as a database backend, or extensions to support alternate querying interfaces, such as SPARQL. Additional plans include the use of the SKOS API as a basis for SKOS validation services. SKOS has a mixture of both semantic constraints that are represented in the schema, along with additional constraints defined in the SKOS reference documentation. For example, constraints on the use of the `skos:prefLabel` property state that every concept can only have one preferred label in a given language. This type of consistency checking can easily be done at the API level and made accessible to users of SKOSEd.

SKOSEd is available as open source²³. We encourage the reader to download and use the application and welcome further comments or suggestions for enhancements.

Acknowledgements. We would like to acknowledge Matthew Horridge for his initial work on the SKOSEd plugin and valuable expertise on the OWL API and Protégé 4. SJ is supported by the Sealife project (IST-2006-027269).

²³ <http://code.google.com/p/skoseditor/>

References

1. Bechhofer, S., Carroll, J.J.: Parsing OWL DL: Trees or Triples? In: Proceedings of the World Wide Web Conference, WWW 2004, pp. 266–275. ACM Press, New York (2004)
2. Bechhofer, S., Volz, R., Lord, P.: Cooking the Semantic Web with the OWL API. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 659–675. Springer, Heidelberg (2003)
3. Bechhofer, S., Yesilada, Y., Stevens, R., Jupp, S., Horan, B.: Using Ontologies and Vocabularies for Dynamic Linking. IEEE Internet Computing 12(3), 32–39 (in press, 2008)
4. Hepp, M., de Bruijn, J.: Gentax: A generic methodology for deriving owl and rdf-s ontologies from hierarchical classifications, thesauri, and inconsistent taxonomies. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 129–144. Springer, Heidelberg (2007)
5. Horridge, M., Bechhofer, S., Noppens, O.: Igniting the OWL 1.1 Touch Paper: The OWL API. In: Proceedings of OWLEd 2007: Third International Workshop on OWL Experiences and Directions (2007)
6. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From *SHIQ* and RDF to OWL: The making of a web ontology language. Journal of Web Semantics 1(1), 7–26 (2003)
7. Lacasta, J., Nogueras-Iso, J., López-Pellicer, F.J., Muro-Medrano, P.R., Zarazaga-Soria, F.J.: Thmanager: An open source tool for creating and visualizing skos. Information Technology and Libraries (ITAL) 26(3), 39–51 (2007)
8. Miles, A.: SKOS: requirements for standardization. In: DCMI 2006: Proceedings of the 2006 international conference on Dublin Core and Metadata Applications, pp. 55–64. Dublin Core Metadata Initiative (2006)
9. National Library of Medicine. Medical subject headings: main headings, subheadings, and cross references used in the index medicus and the national library of medicine catalog, 1st edn. (1960)
10. Peterson, T.: Introduction to the Art and Architecture Thesaurus. Oxford University Press, Oxford (1994)
11. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. J. Web Sem. 5(2), 51–53 (2007)
12. Soergel, D., Lauser, B., Liang, A., Fisseha, F., Keizer, J., Katz, S.: Reengineering thesauri for new applications: the AGROVOC example. Journal of Digital Information 4(4) (March 2004)
13. Soualmia, L.F., Goldbreich, C., Darmoni, S.J.: Representing the mesh in owl: Towards a semi-automatic migration. In: First International Workshop on Formal Biomedical Knowledge Representation (KR-MED 2004) (June 2004)