TEDDY – Thesaurus Editor: Design and Definition Yarn

Magdalena Jitcă, Claudiu Mihăilă,
Anamaria Pradaiş, and Simina Tofan

Faculty of Computer Science,
"Al.I. Cuza" University of Iaşi,
16, G-ral Berthelot Street,
700483 Iaşi, Romania
{magdalena.jitca, claudiu.mihaila,
anamaria.pradais, simina.tofan}@info.uaic.ro

Abstract. In the last years, the semantic Web has increased in size and importance at an inconceivable rate. Along with it, the need for a semantic annotation that can be undertood by machines has developed. This paper introduces TEDDY, a thesaurus and taxonomy editor in the context of the Semantic Web. The design, architecture, implementation and functionality are discussed.

Key words: thesaurus, taxonomy, semantic Web, Jena, SKOS

1 Introduction

A thesaurus is a special kind of vocabulary, as it consists of a collection of terms within a certain domain, which are called concepts. Concepts can have several labels, i.e. different words that are associated with it. These might be definitions, abbreviations, spelling variants, synonyms, translations, and other words that can be used to refer to that term. All this information can be stored along with a concept. Beyond that, a thesaurus is also structured, as it describes the relationships between its concepts. Concepts are arranged in hierarchical and associative relationships. Hierarchical relationships are used to indicate concepts which are narrower and broader in scope, whilst associativity expresses existing connections between concepts.

TEDDY, the Thesaurus Editor: Design and Definition Yarn, is a tool that makes it straightforward to create and maintain thesauri and taxonomies. The user interface (GUI) was designed to be easy and intuitive to use even for people without a Semantic Web background or special technical skills. The use of this editor does not require neither special knowledge, nor extensive training. The system will be described in detail by means of examples and visualisations in the following sections.

The report is structured as follows: sections 2 and 3 describe the vocabularies used within TEDDY, and the Jena framework, respectively. In sections 4 and 5, the

architecture and implementation details are offered, whilst section 6 contains all functionalities available in TEDDY. Finally, the conclusions are drawn and future research directions are stated.

2 Vocabularies

The concepts and the relations between them are annotated semantically using four dictionaries. A very simple example of annotated data is provided in Fig. 1. It can be observed how the different vocabularies are integrated in order to express statements about a subject.

```
<rdf:RDF
xmlns:teddy="http://Teddy#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:skos="http://www.w3.org/2004/02/skos/core#"
xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#">
<rdf:Description rdf:about="http://Teddy#7c4e6560-86a2-4bf0-9ef3-fe04530cad70">
<rdf:type rdf:resource="http://www.w3.org/2004/02/skos/core#CONCEPT"/>
<skos:BROADER rdf:resource="http://Teddy#784f9562-a532-556b-a12c-
a45d70f30ce0"/>
<skos:RELATED rdf:resource="http://Teddy#f3055362-ac92-ba15-4f0c-
4e5625d77a80"/>
<skos:PREFLABEL xml:lang="RO">Tequila Sunrise</skos:ALTLABEL>
<geo:LAT>33.448333</geo:LAT>
<geo:LONG>-112.073889</geo:LONG>
</rdf:Description>
</rdf:RDF>
```

Fig. 1. RDF example.

Each of the four vocabularies are described briefly in the following subsections, explaining their general purpose for which they were originally constructed and their utility in the current project.

2.1 RDFS

The Resource Description Framework Schema (RDFS) [1] is a description language associated to the RDFvocabulary, in order to allow the describing of taxonomies of classes and properties. It also extends the definitions of some elements of the RDF by setting a domain and range for properties, or by relating the RDF classes and properties into taxonomies using the RDFSvocabulary.

2.2 SKOS

The Simple Knowledge Organization System (SKOS) [2] is a formal language for common data modelling, designed for the representation of structured controlled

vocabularies, such as thesauri, taxonomies, or classification schemes. According to the W3C's recommendation:

> Using SKOS, concepts can be identified using URIs, labeled with lexical strings in one or more natural languages, assigned notations (lexical codes), documented with various types of note, linked to other concepts and organized into informal hierarchies and association networks, aggregated into concept schemes, grouped into labeled and/or ordered collections, and mapped to concepts in other schemes.

Two of the classes defined in the SKOS vocabulary are included in TEDDY. The `Concept` class is used for the resources created by users. The `ConceptScheme` class is used to compile these resources into classification schemes.

Furthermore, from the multiple properties the SKOS vocabulary contains, six have been selected to be used within the current project. Their meaning is explained next, and a short example is provided.

- `narrower` – defines a hierarchical relationship between two concepts, the first one being more general, and the second one more specific.
  `<A> skos:narrower <B> .`
- `broader` – defines a hierarchical relationship between two concepts, the first one being more specific, and the second one more general.
  `<A> skos:broader <B> .`
- `related` – defines an associative, non-hierarchical relationship between two concepts.
  `<A> skos:related <B> .`
- `definition` – assigns a definition to a concept.
  `<A> skos:definition "A"@en .`
- `prefLabel` – assigns a preferred label to a concept.
  `<A> skos:prefLabel "colour"@en, "culoare"@ro .`
- `altLabel` – assigns an alternative label to a concept.
  `<A> skos:altLabel "colour"@en, "color"@en .`

These are all properties that are needed in order to create a functional thesaurus, which contains hierarchies, associations, labels and definitions. Other properties may be included in future versions of TEDDY.

2.3 DCT

The Dublin Core Metadata Terms (DCT) [3] is a vocabulary developed by the Dublin Core Metadata Initiative, intended to be used for the description of cross-domain digital information resources, such as text, images, audios, videos, or mixtures of these.

The DCT vocabulary is integrated in TEDDY with the purpose of assigning additional pieces of information to the created concepts. These pieces of information regard the creator of the concept, the date of the creation, other people who have contributed to the concept, and the dates of their contributions.

– `creator` – the original author of the concept.
  `<A> dct:creator "John Doe" .`
– `contributor` – persons who have modified the concept.
  `<A> dct:contributor "Jane Doe" .`
– `date` – the date of the concept's creation.
  `<A> dct:date "2010-01-31" .`
– `modified` – the date of the contribution.
  `<A> dct:modified "2010-02-01" .`

By using these properties of the DCT vocabulary, TEDDY has the ability of maintaining the history of the concepts in the thesaurus.

2.4 WGS84

The World Geodetic System (WGS) [4] is a standard for use in cartography, geodesy, and navigation. The latest version is WGS84, which dates from 1984, but which was revised in 2004. Furthermore, this version is the reference coordinate system used by the Global Positioning System.

Two properties defined in this vocabulary are used within TEDDY, in order to express the location of a concept on the surface of Earth.

– `lat` – the latitude at which the concept is found.
  `<A> geo:lat "47.173907" .`
– `long` – the longitude at which the concept is found.
  `<A> geo:long "27.574739" .`

These two pieces of information are used by TEDDY for the inclusion of a map pinned at that precise location.

3 Jena

Jena[1] is an open source Java-based framework used for the development of web applications. It offers tools for RDF, RDFS and OWL processing, as well as ARQ and SPARQL query engines. In addition, it provides two storage engines (TDB and SDB) to cope with the ARQ engine. Jena is available for both Linux and Windows distributions, and is currently at version 2.6.2.

The Jena API has been defined in terms of interfaces in order to allow developer applications to be independent of the implementations. For this purpose, the core package of the distribution (`com.hp.hpl.jena.rdf.model`) has been designed to contain the interfaces useful for representing models, resources, properties, literals, statements and all the other key concepts of RDF, as well as a `ModelFactory` for creating models.

Jena supports three types of operations on RDF models: union, intersection and difference. This basically means that a model can result after applying one of the above mentioned operations on two input models.

[1] http://jena.sourceforge.net/

RDF has special representations of collections of objects, which are known as containers. This type of resources contains either literals (property values) or other resources. Jena works with three kinds of containers: (i) `Bag` is an unordered collection, (ii) `Alt` is a collection of unordered elements aimed to be used alternatively, and (iii) `Seq` represents ordered collections. Jena also provides tools for container management and ensures, for example, that the first element is always labeled with the `rdf:_1` index, or that there are no unused indexes in the `[1, sizeOfCollection]` range.

With regard to vocabularies, Jena operates with multiple, widely used ones, such as FOAF, VCARD, RDF, and DC. All of its vocabularies are included in the `com.hp.hpl.jena.vocabulary` package. The RDF vocabulary has a predefined namespace, whilst all other vocabularies are assigned to the default namespace. However, Jena provides several ways of improving the access to RDF documents by controlling the namespaces used with its prefix mappings. In addition, Jena provides a tool named "schemagen", which is used to convert an OWL, DAML+OIL, or RDFS vocabulary into a Java class that contains static constants for the named classes, properties and instances of the input vocabulary. Furthermore, the output can be formatted by means of a template.

We have decided to integrate Jena in our project primarily for performance and scalability reasons. Jena is highly scalable even when dealing with models containing 200 million triples, and does not misbehave with regard to memory usage. Nevertheless, the current performances can be improved by adopting query optimisation techniques. Furthermore, due to its Java based framework, Jena is also very easy to incorporate in applications developed using the same language. Moreover, it offers a stable, well-documented, and easy to use API.

4 Architecture

The main purpose of TEDDY is to reduce the superfluous work, especially for those users who are not acquainted with the Semantic Web.

The TEDDY system is built following the Model-View-Controller architectural pattern, in order to isolate the business logic from the user interface. Thus, TEDDY results in an application in which it is easier to modify either the visual appearance of the application, or the underlying business rules without affecting the other.

The architecture of TEDDY, depicted in Fig. 2, aims to be modular and service oriented. By following this approach, it is feasible to externalise some services which already exist, such as persistent storing, or image generation. Another advantage is that it is easier to add new services to the system once they are necessary. Furthermore, a service-oriented architecture ensures a high degree of cross-platform interoperability.

The data to be processed – in this case, the RDF statements – are represented in an internal model. Views and editors access the model via a controller class. Changes at either end of the application are propagated to the other.

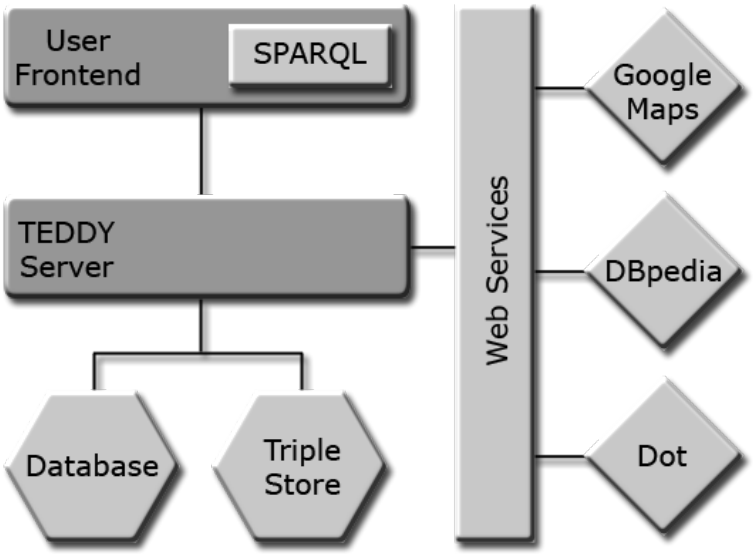

Fig. 2. Architecture of the TEDDY project.

5 Implementation

The TEDDY project is implemented using the Java language, due to its platform independency, compatibility with the XML document management libraries, and moderate requirements when installed on machines. The chosen software development environment is Eclipse[2]. This decision is due to the workbench's robust infrastructure and extensible plug-in system, which makes it possible to efficiently design and implement applications for various purposes.

The selected application server is JBoss Application Server[3], due to its robustness and scalability features. Moreover, the server is freely available under a GPL license.

With regard to version control, TEDDY was developed using the freely available SVN service available at Google Code[4].

The class diagram used to model the application is included in Fig. 3. As it can be observed, the `Concept` class contains all the properties stated in section 2. Moreover, users are able to set a default language for their work and may share (or not) the projects they have created with others. The classes `ThesaurJavaMethods` and `ThesaurRDFMethods` are the classes which link the

[2] http://www.eclipse.org/
[3] http://www.jboss.com/products/platforms/application/
[4] http://code.google.com/p/semantictteddy/

RDF statements to Java objects, in order to facilitate the interaction with the user.

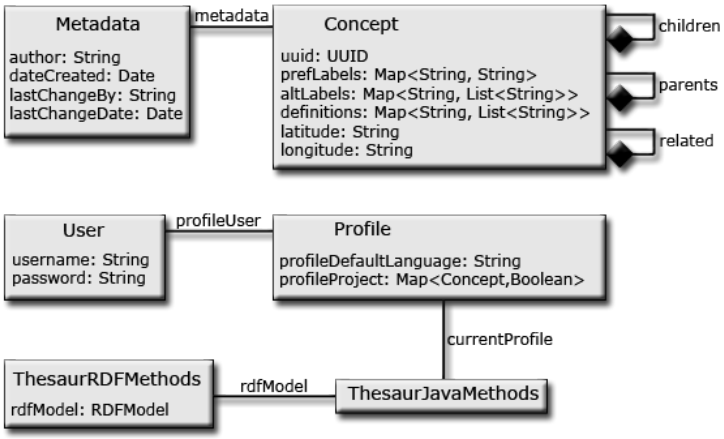

Fig. 3. The class diagram for TEDDY.

6 Functionality

There are many functionalities included in TEDDY, in order to ensure a forthright and proficient user experience. Different file formats accepted for import and export, visualisation of concepts as trees, as property-value lists, or in graphics, and a SPARQL endpoint are explained in the following subsections.

Fig. 4 shows TEDDY's graphical user interface. The information about the project is organised and displayed into two main areas:

- concept hierarchy tree – displays the hierarchical structure of the concepts in the project. Click on the boxes to select a thesaurus or a concept. For details see section 6.2.
- concept details view – this part of the interface shows all the details stored for the selected concept. The tab bar can be used to switch between different views.

The navigational items seen in Fig. 4 are:

- main menu bar - for managing the projects and documents, usage of tools and options.
- tab bar - for each selected concept the user can choose from a number of views displaying different information or alternative ways to explore the concept's details.

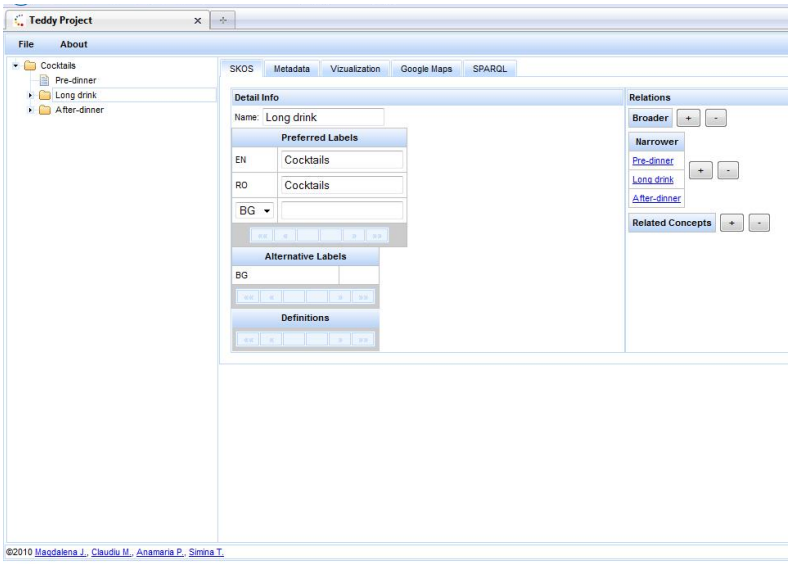

Fig. 4. The graphical user interface of TEDDY.

6.1 Import and export

TEDDY supports importing and exporting existing thesauri and taxonomies from several formats, such as RDF/XML, RDF/XML-ABBREV, N-TRIPLE, N-3and TURTLE. All the import and export operations are available via the Jena API by means of the core `Model` interface.

The difference between the previously mentioned RDF/XML formats is only visible at exporting content and can be measured in terms of readability vs. efficiency. The first one produces efficiently regular output, but it is not readable by humans. In contrast, RDF/XML-ABBREV produces readable output without focussing on efficiency. However, if one wants to optimise the speed of writing RDF/XML, it is suggested to turn off any URI processing.

The N-TRIPLE readers and writers implement the RDF Core N-Triples language[5], and do not offer the posibility of customising the configuration.

On the contrary, the N-3 output can be furthermore specified as: N-3PP, N-3PLAIN, or N-3TRIPLE, acccording to Tim Berners-Lee's implementation of the N-3 language[6]. The N-3 pretty printer (N-3-PP) is enabled by default and provides the possibility to control the output by means of properties.

The TURTLE[7] readers and writers are very similar to the N-3 ones, and use the character set from the XML namespaces.

[5] http://www.w3.org/TR/rdf-testcases/#ntriples
[6] http://www.w3.org/2000/10/swap/Primer.html
[7] http://www.w3.org/TeamSubmission/turtle/

6.2 Tree visualisation

The main purpose of TEDDY is to create taxonomies and thesauri in an efficient and effortless manner. Thus, due to the properties specific to these types of classifications (such as narrower and broader), hierarchies of concepts result, and they are best represented as trees. A depiction of a tree of concepts is included in Fig. 5.

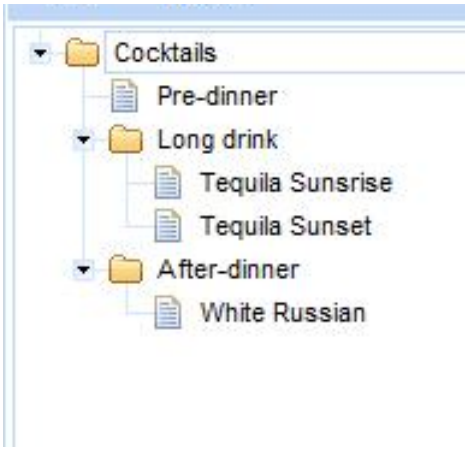

Fig. 5. TREE

The top node of the tree represents the project. The project's childen nodes are the concept schemes, or individual thesauri. The concept schemes are composed of hierarchies of nodes, which correspond to the concepts in the thesaurus.

6.3 Property-value visualisation

The properties associated to concepts can be assigned values or can be edited in this tab.

Data All properties defined for a concept in the TEDDY project are displayed and may be edited in the data area, depicted in Fig. 6.

In the relations panel, concepts may be linked to other concepts using the `skos:broader`, `skos:narrower` and `skos:related` relations. Furthermore, new concepts can be created here, as narrower concepts of the current one. It is important to be noted that this is the only way new concepts can be introduced in the model.

Neighbouring the relations panel is the data properties panel. All properties that accept literals as their range are given here. The user is able to add new values to the properties, and select one of the 23 official European languages for

the text. Moreover, the user may edit or remove property values according to their needs.

The preferred label is the main word or phrase that is used by humans to identify the concept. Each concept must have a preferred label, but not more than one for each language. On the other hand, alternative labels are other words or phrases used to refer to this concept. This includes synonyms, acronyms, abbreviations, spelling variants, or irregular plural/singular forms.

Furthermore, the definition property, which is a description of the meaning of the concept, can be found, and it is similar to an alternative label – it may appear multiple times for one language.

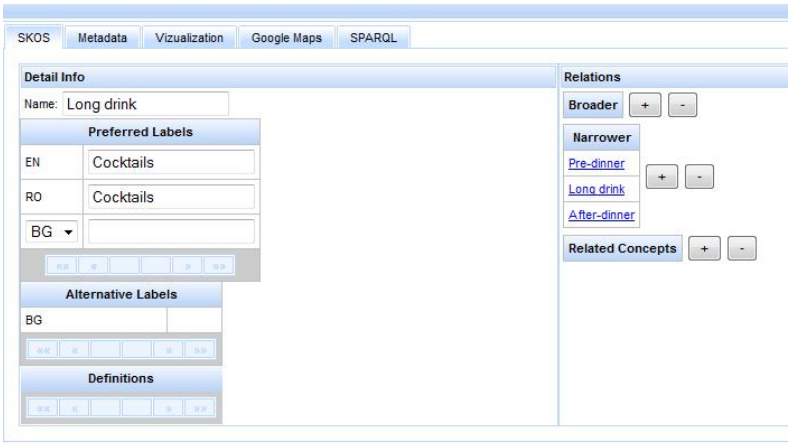

Fig. 6. SKOS

Metadata In the metadata section, the information regarding the author and contributors are displayed, such as in Fig. 7. The data included here is not editable, since it is automatically generated when users perform actions on concepts.

6.4 Graphic visualisation

The information associated with a concept can be visualised also graphically. TEDDY provides images for concepts, showing their relations to other concepts and, if applicable, their location on Earth. Both types of images are detailed in what follows.

Relations The relations that exist between concepts can be better understood by looking at an image rather than trying to understand the RDF statements.

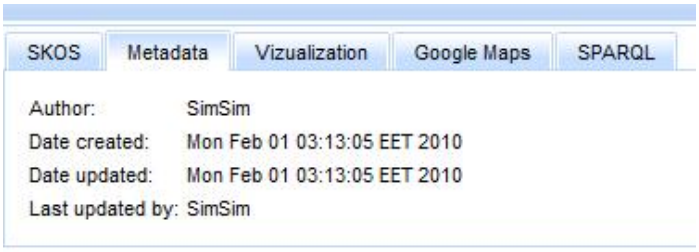

Fig. 7. META

For this purpose, TEDDY is able to include a visual representation of a concept's relations. The respective image is generated by firstly transforming the RDF statements into DOT code, such as in Fig. 8. Afterwards, the code is fed to the *dot* program, which is part of GraphViz[8], a graph visualisation software. Then, the program creates an image, an example of which is included in Fig. 9.

This method was chosen due to the versatility of *dot*, expressed in the multiple image file types it can generate, and in the ease with which complex graphs are depicted.

```
digraph G {
node [shape = doublecircle]; "Tequila Sunrise";
node [shape = ellipse]; "Long drink";
node [color = "#000000", fillcolor = "#EEEEEE", style = filled]; "Tequila Sunset";
"Astronaut Sunrise";
"Tequila Sunrise" -> "Long drink" [label = "B", color = "#000000"];
"Tequila Sunrise" -> "Tequila Sunset" [label = "R", color = "#999999"];
"Tequila Sunrise" -> "Astronaut Sunrise" [label = "R", color = "#999999"];
}
```

Fig. 8. Transformation of RDF statements into DOT code.

Location In the case of concepts which may have a localisation on Earth, TEDDY includes a map of the specific place. If the `geo:latitude` and `geo:longitude` properties are set, the map is obtained from Google Maps[9], using the available API and displayed.

By using such a feature, TEDDY diminishes the chances of confusion in the case of ambiguous concepts, and increases the level of available information for the user.

[8] http://www.graphviz.org/
[9] http://maps.google.com/

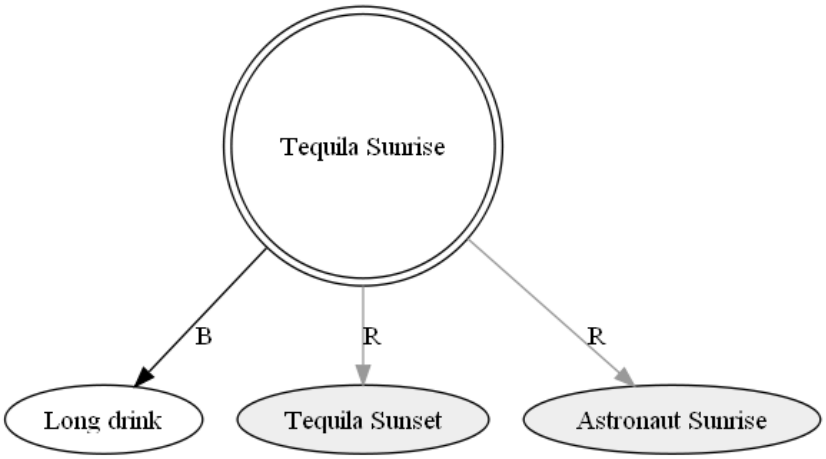

Fig. 9. Rendering of RDF statements generated by *dot*.

6.5 SPARQL endpoint

TEDDY also provides a SPARQL[10] query interface, which allows the user to retrieve information regarding within existing ontologies. This is particularly useful for distributed queries, where one can incorporate data from various sources. Users can either write their own queries, or choose from the ones provided by our application. The predefined interrogations stored in our system query both DBpedia and the RDF models defined by the user. The results are displayed on the main screen in real time. We have provided SPARQL samples for what we consider frequently queried items, with the scope of providing help for less experienced users. For example, the query in Fig. 10 searches for the concepts whose preferred label starts with a *C*.

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT distinct ?concept ?label WHERE {
?concept skos:PREFLABEL [ ]
skos:PREFLABEL ?label
FILTER (regex(str(?label), '^c', 'i'))
}
```

Fig. 10. SPARQL query retrieving all distinct concepts whose preferred label starts with *C*.

SPARQL is supported in Jena by means of the ARQ module. The ARQ query engine can parse queries expressed not only in SPARQL, but also in RDQL or its own internal query language. ARQ is under active development and is not yet part

[10] http://www.w3.org/TR/rdf-sparql-query/

of the standard Jena distribution, nevertheless it is available as a self-contained download.

7 Use case

The first action to perform in order to have access to the projects management is user login. The persons should register by using their username and their stored password. After a successful login, one can have access to the previously stored projects or to creating a new one. In order to open an existing project select 'Load Project' from the menu and select one from the presented list.

To create a new project, one must click on 'New Project', which brings up the project creation dialogue. This action opens a window where one can enter the name, subject and other information about the project, as well as select the language for the project. The user scrolls through the list or type the first letters of your desired language to select the 'default language' for the project. It will be used as default for displaying the labels of concepts. The user can add more languages by selecting them and clicking on the arrow pointing to the right and remove them with the arrow pointing to the left. After pressing 'Create Project' the user will be ready to create thesauri and new concepts or import an existing thesaurus. When they finished their work, they can log out by selecting the 'log out' option from the menu.

In order to create new concepts users just have to click the plus icon in the 'Narrower' area of a selected concept. All information for a selected node (concept or thesaurus) can be edited and added inline. Clicking the plus + icon lets you add new labels or definitions. To delete information press the minus - icon. Preferred labels cannot be deleted (one has to edit them or delete the whole concept) since exactly one preferred label per concept (and per language) must exist.

The same goes for the editing of relations between concepts. To create a new concept related to the current one via narrower, broader, or related one uses +. If the concept one wish to link to already exists, a new concept will however be created because TEDDY supports different concepts with the same label. To remove a relationship between concepts use the icon -. Clicking it will delete relations, and not the concepts themselves, thus disconnecting concepts that were previously related to each other via narrower, broader, or related. However, a concept should only remain in the thesaurus, if it is connected to a least one parent concept. This means, removing the last broader relationship of a concept would not result in the deletion of this concept. For example, if the concept 'Tequila Sunrise' has no broader relationship to any other concept in the project, removing it from 'Long Drinks' would not delete this concept entirely, thus TEDDY offers the option of reusing the concept at a later time.

8 Conclusions

We have introduced TEDDY, a web application that permits the editing of taxonomies and thesauri for the Semantic Web. The overall architecture and implementation were briefly discussed, whilst the multiple functionalities were detailed and exemplified.

There are many possible future developments, either to increase the scope of the modelled knowledge, or to enhance the visualisation methods. For instance, more vocabularies might be added (e.g., FOAF, DOAP), and the user could be allowed to define their own properties to relate concepts. Concept merging is an interesting idea, since state of the art methodologies are not yet able to perform ontology merging perfectly. Furthermore, machine learning techniques can be applied in order to create a tag recommender, or to autocomplete all the labels in other languages, provided that one is given. Moreover, a tag cloud generator service could be included to show the importance of concepts.

9 Acknowledgements

We are grateful to Andreas Blumauer, who has kindly granted us access to PoolParty[11], and allowed us to test it.

References

1. W3C: RDF vocabulary description language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/ (10 February 2004)
2. W3C: SKOS simple knowledge organization system reference. http://www.w3.org/TR/skos-reference/ (18 August 2009)
3. Dublin Core Metadata Initiative: Dublin Core metadata terms. http://dublincore.org/2008/01/14/dcterms.rdf (14 January 2008)
4. W3C: Basic geo (WGS84 lat/long) vocabulary. http://www.w3.org/2003/01/geo/ (20 April 2009)

[11] http://poolparty.punkt.at/PoolParty/