A Tool for Collaborative Ontology Development for the Semantic Web

Arttu Valo, Eero Hyvönen, Ville Komulainen
Helsinki University of Technology and University of Helsinki
Laboratory of Media Technology
P.O. Box 5500
FIN-02015 TKK, FINLAND
FirstName.LastName@helsinki.fi

Abstract:

We present a national ontology library development framework ONKI under development in Finland. ONKI's main goal is to support collaborative development and re-use of interdependent ontologies. It features change management and versioning of ontologies as well as a browser component which provides the ontology search and utilization services as Web Services.

Keywords:

Semantic web, re-use, ontologies, ontology library, collaboration, web services.

1. Introduction

Growing interest and need for both human and machine understandable information on the www is driving the research and development in Semantic web ontologies. Ontologies contain information about concepts and their relations. For example, a banking ontology would contain information of the concepts of different types of accounts, transactions, currencies, and how transactions have debit and credit accounts and amounts of currency as their properties.

Unfortunately, current support systems for ontology development are limited especially in the areas of interoperability and re-use. Reminiscent of the older, non-distributed artificial intelligence and knowledge representation, these tools generally lack the core nature of web-related development – interoperability. Thus, semantic web faces the risk of becoming an archipelago of separated islands instead of a unified web of re-using and expanding on existing systems. Ontology library systems are proposed as a solution for the lack and difficulty of re-use [1].

2. Ontology Development Process

Ontology development typically follows an iterative model – an initial publication followed by a long maintenance time. During maintenance, varying numbers of changes are introduced and new versions published.

Both human and machine understandable ontology development resembles that of software modeling and development, with modeling problems and releases. Thus, we can use approaches for distributed development used in software development. However, the hypertextual nature of semantic web ontologies requires us to include assisting mechanisms to address issues beyond file- and fileset-level structures used in common software development platforms. For example, the use of revision control systems such as CVS[1] for semantic web ontologies [2] must be considered carefully since they are not designated for graphed material such as RDF, but typically for linear text-files.

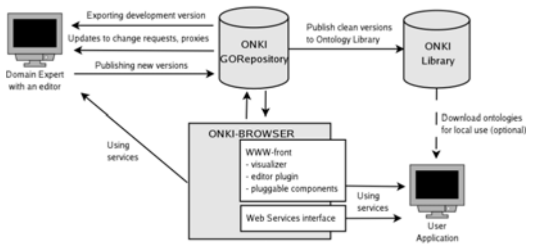

Figure 1: ONKI development and publishing process

The ONKI architecture and publishing process is depicted in Figure 1. The three core

[1] http://www.cvshome.org

components of the system are the development repository (ONKI GORepository) for fine-grain access to any versions of the ontologies, the public ontology library containing the sets of versions of published interrelated ontologies (ONKI Library), and the browsing service (ONKI Browser) for using the ontologies.

ONKI separates the development process into two major parts: the *development loop* (cf. the arrows between Domain Expert and ONKI GORepository in Figure 1) and the *publishing push* (cf. the bottom arrow from Domain Expert to ONKI GORepository and the arrow from there to ONKI Library in Figure 1). The development loop is based on exporting an ontology with versioning metadata and *proxies* in place to protect the distributed development.

Notification and changes are occasionally polled from the development repository, especially for seeing whether there have been affecting changes in other ontologies. Pull is used to download the affecting changes made in related ontologies because it has been identified as better than other mechanisms for keeping distributed ontology copies and development synchronized [4].

All changes, e.g., the introduction of a new concept in a subsumption hierarchy, are documented as instances of an RDF change ontology and are stored as metadata of the ontology development version when editing an ontology. Concepts from related ontologies can be imported by a proxy-mechanism that creates a copy of the borrowed concepts and keeps track of their origin.

Having a securely contained development copy of imported concepts allows the ontology developer to focus on her modeling work without constant worries of changes in dependencies with other ontologies in the library. Furthermore, the system also maintains dependency information in the concept’s home ontology so that the consequences of making changes are apparent to that ontology’s editor.

Once an ontology editor decides that the current development version is mature enough for publishing, the development changes are sent to the development repository. The development repository keeps track on individual concepts and ontologies that contain them. Both concepts and ontologies are versioned.

When publishing an official version of the ontology, the publishing push is automatically activated at the development repository. In publishing push, the development metadata that was created during development is separated from the clean ontology version. This results in two published packages: 1) A cleaned-up, readily-usable ontology — in our case an RDF Schema file. 2) An RDF file containing the change instances from the previous version to this just published latest version. This change metadata can be used when the developers of other ontologies want to upgrade their own ontologies to meet the changes or by other users who cannot or do not want to download the new version as a whole.

3. Development Repository Functionality

The development repository maintains metadata of versions for concepts and ontologies, and tells to what ontologies concepts belong to. In addition, it keeps track of change requests, i.e., changes not made by the owner of the ontology. The metadata of changes during ontology development is based on two mechanisms: changes and proxies.

1.1 Describing Changes

There are two kinds of change descriptions in ONKI. First, metadata of concept and ontology changes in the edited ontology is maintained. Second, the system also records change requests imposed by changes made in other ontologies or fed in through the feedback channel for changes. For example, if the original version of a borrowed concept is moved from an ontology to another, then the new origin information should be updated in all ontologies using the concept.

Knowing the change history of ontologies is important in synchronizing ontology development of related ontologies and in keeping the versions interoperable with each other. In PROMPTdiff [3], ontology changes are identified automatically by comparing two versions and then deducing the changes. Since this approach cannot necessarily identify and describe all changes accurately, we decided that the editor should record and explain the changes explicitly during the development process in terms of change metadata.

Change requests differ from changes only in that they have been imposed by changes made in other related ontologies. Pending change requests are visible in the development repository through the ONKI Browser. The editor can then decide whether to turn the request into an official change, to remove the request entirely, or to remain undecided and let the change request lay in the development repository.

ONKI system can also accept ontologies from editors who do not produce full track of change instances. This does, however, prevent publishing meaningful change sets and will cause a breach in the changes chain for both the ontology and its concepts.

1.2 Using Proxies

Proxies are a crucial mechanism for separating individual ontologies for distributed development. A proxy is a local representation of a remote entity — in our case an ontology concept. When an ontology is exported for development, its references to other ontologies within the development repository are replaced by proxies. Thus, instead of referring to external concepts, the references are being made to temporarily inserted proxies.

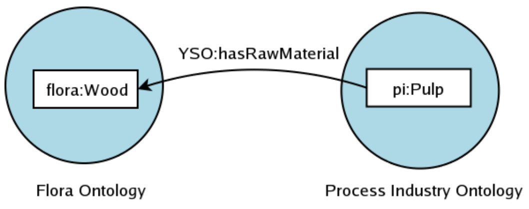

Figure 2: Simple interdependent ontologies

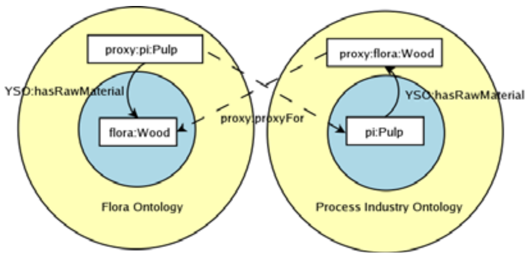

Figure 3: Simple ontologies with proxies providing isolated development

Figure 2 exhibits a simple example of a dependent pair of ontologies. Any change in Flora ontology will directly affect the depending Process Industry ontology. For example, if the Wood class is divided in more detail into different types of trees, only part of them are likely to be good ingredients when making pulp. Furthermore, Wood might not really belong to Flora ontology – it could better belong into some Natural Materials ontology while the actual species of trees would be listed in Flora ontology. Any of these changes would affect the Pulp class in Process Industry Ontology. A detailed list of change effects can be read from [5].

Figure 3 presents the situation in development versions after the introduction of proxies. Both ontologies have a protective one-arch-wide cloud of proxies which hide the direct references and instead point to the versions at the creation time of the proxy. When proxies are created to both ontologies, the dependency is visible for editors of both ontologies. The editor software’s knowledge of the proxy ontology limits the visualization of the dependencies, but the proxy-system should also work on more generic editors.

Technically, when we create a proxy, we create a temporary copy with a URI in the proxy-namespace and give it at least one property: the proxyFor RDF literal, which has the original, proxied URI as a string value. The reason for it not being a RDF resource is to prevent current editors from traversing through it during development. The concept's home ontology (Flora ontology in Figure 3) will have a similar proxy of the other concept to keep its editor knowledgeable of having depending references.

A proxy can have any range of properties of the entity it represents. During the development time the editor is free to modify its properties just as those of her own concepts'. If and when the properties of a concept are changed in its own home ontology, polling can be used to update to the latest version. Thus, the developer of an ontology can have a isolated development version and work with it independently of changes elsewhere.

It is the duty of the development repository import and export functions to retain unique references to the proxies and versions during development. The development version’s interdependencies are not limited to existing proxies created by the development repository’s export functions. New concepts from other ontologies can be “taken into use” into a development version through the services of ONKI Browser. In practice, the services will create proxies.

Upon publishing, temporary metadata such as proxies are removed, URIs are reverted to point to the actual entities, and official change set is made publicly available with the new version. Thus, proxies with no references left will be automatically cleaned up.

4. ONKI Browser and Interfaces

ONKI Browser is used for illustrating, finding, and importing concepts from the ONKI system ontologies (cf. the arrow Service utilization in figure 1). It consists of three components: 1) Connector to ontology repository that has utilities for knowledge-base information retrieval processes. 2) Visualizer for the semantic data, collecting the data from Connector according to parameters given. 3) Web Service interface for intelligent agents and applications. This interface is as a wrapper to the Connector providing access to ontological information.

An ontology library such as ONKI can contain lots of separate ontologies. A domain expert editing one ontology with an ontology editor, such as Protege-2000, typically can not see the other related ontologies stored in the library. Thus, it is essential to have a tool for gaining a clear perception of the ontology library as a whole. Although visualizing complex semantic data in HTML is challenging, HTML is a good platform since there is no need for any additional software installation or plugins for the end user.

Therefore ONKI Browser is implemented by server side programming providing visualization of ontological data for all devices equipped with an ordinary HTML-browser. Interface also offers search engines an option of querying ONKI and look for synonyms or closelely related concepts matching the specified search-criteria and later guide Search engines user towards the answers he/she was really looking for.

Machine understandable interfaces are required for sharing and using knowledge stored in ontologies. External applications can use ONKI by the Simple Object Access Protocol (SOAP) over HTTP. SOAP enables applications to connect to ONKI and execute queries defined by SOAP-methods on the knowledge base. This mechanism gives, for example, the possibility to annotate external application data with the concepts in ONKI by implementing SOAP-calls to the server. ONKI Browser can be used for finding and selecting the annotation concepts. The benefits of such a centralized ontology service are clear: Firstly, external applications can reuse the ONKI Browser functionality. Secondly, concept labels and especially the underlying complex URIs can be imported easily into applications. Thirdly, the service on the web always contains up-to-date versions of the ontologies.

5. Discussion

The need for ontologies is clearly visible, but the tools for the development, management, and publishing ontologies are just being developed. Limited interoperability between knowledge editors and management systems and a generic fragmentation of approaches is still a major obstacle for developing interdependent ontologies. ONKI system tries to alleviate these by providing a non-intrusive, interoperable framework for distributed collaborative development, straightforward publishing, and web service based usage of ontologies.

References

1. Y. Ding, D. Fensel. Ontology library systems: the key to successful ontology reuse. In The first Semantic Web symposium (SWWS1), Stanford, USA, July 29-August 1 2001.
2. T. Korpilahti, E. Hyvönen. An Architecture for Collaborative Ontology Library Development. Paper, University of Helsinki, 2004.
3. M. Klein. Change Management for Distributed Ontologies. PhD Thesis, SIKS, The Dutch Graduate School for Information and Knowledge Systems, 2004.
4. L. Stojanovic. Methods and Tools for Ontology Evolution. PhD Thesis, Universität Karlsruhe, 2004.
5. N. Noy, M. Klein. Ontology evolution: Not the same as schema evolution. Knowledge and Information Systems, 5, 2003.