Proceedings of the First Workshop on Ontology Repositories and Editors for the Semantic Web (ORES2010)

7th Extended Semantic Web Conference Hersonissos, Crete, Greece, May/June 2010

edited by Mathieu d'Aquin, Alexander García Castro, Christoph Lange, and Kim Viljanen

May 31, 2010

Preface

The number of ontologies being built and made available for reuse has increased steadily in the last few years. Semantic Web search engines such as Swoogle and Watson currently index several tens of thousands of them. While being a foundation for the Semantic Web, this new environment where ontologies are shared and interlinked online also poses new challenges; fostering thus a number of research projects aiming to understand, amongst others, ontology reuse, storage, publication, retrieval and modularization. Within the context of the Semantic Web ontology repositories should facilitate the sharing and discovery of reusable ontological components (entire ontologies or portions of them).

Illustrating the importance of the problem, significant initiatives are now emerging. One example is the Open Ontology Repositories (OOR) working group set up by the Ontolog community. Another example is the Ontology Metadata Vocabulary (OMV) Consortium, addressing metadata for describing ontologies. Despite these initial efforts, ontology repositories are hardly interoperable. Although sharing similar aims (providing easy access to Semantic Web resources), they diverge in the methods and techniques employed for gathering these documents and making them available; each interprets and uses metadata in a different manner. Furthermore, many features are still poorly supported, such as modularization and versioning, as well as the relationship between ontology repositories and ontology engineering environments (editors) to support the entire ontology lifecycle.

The growing number of online ontologies makes the availability of ontology repositories, in which ontology practitioners can easily find, select and retrieve reusable components, a crucial issue. The recent emergence of several ontology repository systems is a further sign of this. In order for these systems to be successful, it is now critical to achieve interoperability between ontology repositories, through common interfaces, standard metadata formats, etc.

To address these needs and to provide a forum for researchers and developers to discuss features and exhange ideas, the Ontology Repositories and Editors for the Semantic Web (ORES 2010) workshop was held on Crete on May 31st, 2010, with results documented in this proceedings publication. We thank all participants of the workshop, whose excellent presentations and input inspired insightful discussions. We also thank the program committee for their valuable work in reviewing the articles. Finally, we thank the organizers of the Extended Semantic Web Conference (ESWC 2010) for the practical arrangements.

Mathieu d'Aquin, The Open University
Alexander García Castro, Bremen University
Christoph Lange, Jacobs University Bremen
Kim Viljanen, Aalto University

Contents

Programme

09:00–10:30 Session 1

09:00–09:10 Opening Ceremony
Mathieu d'Aquin, Alexander García Castro, Christoph Lange, Kim Viljanen

09:15–10:00 Keynote
Nigam Shah, Stanford Center for Biomedical Information Research

10:00–10:30 **Demo Presentations**

SOBOLEO – A Repository for Living Ontologies
Simone Braun and Valentin Zacharias

Collaborative Metadata Editor Integrated with Ontology Services and Faceted Portals
Jussi Kurki and Eero Hyvönen

Linked Open Ontology Services
Kim Viljanen, Jouni Tuominen, Mikko Salonoja and Eero Hyvönen

A User Interface for Ontology Repositories
Jouni Tuominen, Mikko Salonoja, Kim Viljanen and Eero Hyvönen

OWLGrEd: a UML Style Graphical Editor for OWL
Jānis Bārzdiņš, Guntis Bārzdiņš, Kārlis Čerāns, Renārs Liepiņš and Artūrs Sprogis

ORE-MP: Ontology Reasoning Engine for Molecular Pathways
Renato Umeton, Beracah Yankama, Giuseppe Nicosia and C. Forbes, Jr. Dewey

CONSISTOLOGY: A SEMANTIC TOOL TO SUPPORT ONTOLOGY EVOLUTION AND CONSISTENCY
Najla Sassi, Wassim Jaziri and Faiez Gargouri

10:30–11:00 **Coffee Break (poster/demo session open to visitors)**

11:00–13:00 Session 2

11:00–11:15 **Poster/demo session continues**

11:15–13:00 **Regular talks (full talks: 15 minutes + 5 minutes discussion; short talks: 10 minutes + 5 minutes discussion)**

11:15–11:35: Previewing OWL Changes and Refactorings Using a Flexible XML Database
Christoph Lange and Vyacheslav Zholudev

11:35–11:55: On the Use of Transformation and Linked Data Principles in a Generic Repository for Semantic Web Services
Barry Norton and Mick Kerrigan

11:55–12:15: iServe: a Linked Services Publishing Platform
Carlos Pedrinaci, Dong Liu, Maria Maleshkova, David Lambert, Jacek Kopecky and John Domingue

12:15–12:30: Context-aware access to ontologies on the Web
Patrick Maué, Alejandro Llaves and Thore Fechner

12:30–12:45: Ontology Recommendation for the Data Publishers
Antoine Zimmermann

12:45–13:00: Semantic Interoperability Framework for Estonian Public Sector's E-Services Integration
Kalle Tomingas and Martin Luts

13:00–14:30 **Lunch**

14:30–16:00 Session 3

14:30–14:55 **Best paper**

14:30–14:55: Ontology Repositories with Only One Large Shared Cooperatively-built and Evaluated Ontology
Philippe Martin

14:55-16:00 **Regular talks (full talks: 15 minutes + 5 minutes discussion; short talks: 10 minutes + 5 minutes discussion)**

14:55–15:15: Using extended metadata model OMV and metrics in OntoLP Portal
Anderson Bestteti, Larissa Freitas and Renata Vieira

15:15–15:30: Ontology Repository for User Interaction
Martins Zviedris

15:30–15:45: Evaluation Framework for Ontology Development and Management Methodologies
Dionisis Kehagias, Dionysia Kontotasiou and Dimitrios Tzovaras

15:45–16:00: A pan-European repository: SEMIC.EU as the point of reference for eGovernment ontologies
Klaus Reichling, Martin Luts and Renke Fahl-Spiewack

16:00–16:30 **Coffee Break**

16:30–18:00 Session 4

16:30–17:15 Keynote
Ken Baclawski, Northeastern University, Boston / OOR Initiative

17:15–17:50 Open panel discussion about OOR (Open Ontology Repository) with "lightning" inputs from the participants

17:50–18:00 Concluding Remarks

20:45– ORES Social Event

SOBOLEO – Editor and Repository for Living Ontologies

Simone Braun and Valentin Zacharias,

Forschungszentrum Informatik, FZI
Haid-und-Neu Strasse 10-14
76131 Karlsruhe, Germany
{braun,zach}@fzi.de

Abstract. SOBOLEO is a web based system that enables groups of people to collaboratively develop and use SKOS ontologies and semantically organized information spaces. SOBOLEO supports the development and refinement of living ontologies – i.e. ontologies that are never finished and that are used and developed at the same time. It offers tools to edit the SKOS ontology used and the information space. It also offers interfaces for remote applications to be notified of changes and to change the ontology itself.

Keywords: Ontology engineering, SKOS, social software, collaboration, semantic annotation, Ontology Maturing, SOBOLEO

1 Introduction

SOBOLEO[1] is a system that enables groups of people to jointly structure information (documents and experts) in a domain. SOBOLEO's goal is to serve as a repository for 'living ontologies' according to the principles of Ontology Maturing [1]. This theory understands ontology development as a continuous and collaborative process embedded and interwoven with actual usages processes; e.g. the users can directly change the ontology when finding it deficient during use of the semantic search. All modeling can be done while the application is used (i.e. no large up-front investment in ontology engineering) and even partial formalizations are used immediately to improve user experience [2].

In this paper, we start with a short overview of use cases for SOBOLEO before introducing design principles and core concepts. In the next section we give an overview of the main functions of SOBOLEO by looking at ontology editing, document and person annotation, semantic search, and browsing. Finally we conclude with a discussion of related systems.

[1] An installation of the SOBOLEO system is publicly available at http://tool.soboleo.com. Users can try out the system in one collaboration space that is open to edits from anyone. SOBOLEO is completely web based – no installation or registration is required.

2 Use Cases

Example use cases for SOBOLEO are:

- An HR department organizing data about available experts. Here SOBOLEO supports an HR department in maintaining a competency ontology and using this ontology to keep track of - and search within - the database of experts. Note that the competency ontology is never really finished – as new possible skills are emerging all the time.
- A learning course that is jointly developing an understanding of a domain, creating an ontology and interlinking it with relevant documents. SOBOLEO supports this use case through the management of both the ontology and the relevant documents. Note that here the ontology is also evolving throughout its use – always reflecting the current knowledge of the course participants.
- A group of expert jointly collecting the state of the art in a scientific domain (with links to both documents and experts). Here, too, the ontology will constantly evolve to reflect both the knowledge of the experts as well as the current scientific consensus.

3 Architecture & Implementation

SOBOLEO is fundamentally organized around the concept of **Collaboration Spaces**. A Collaboration Space is the virtual space in which collaboration between people with a shared goal takes place. Each Collaboration Space has one SKOS ontology and may have information about documents and experts that are annotated with this ontology. All information in a space is jointly edited by the members of this space (spaces can also be configured to allow anonymous users to read and write). Each SOBOLEO installation supports an arbitrary number of collaboration spaces and these are (except for user data) completely independent of each other.

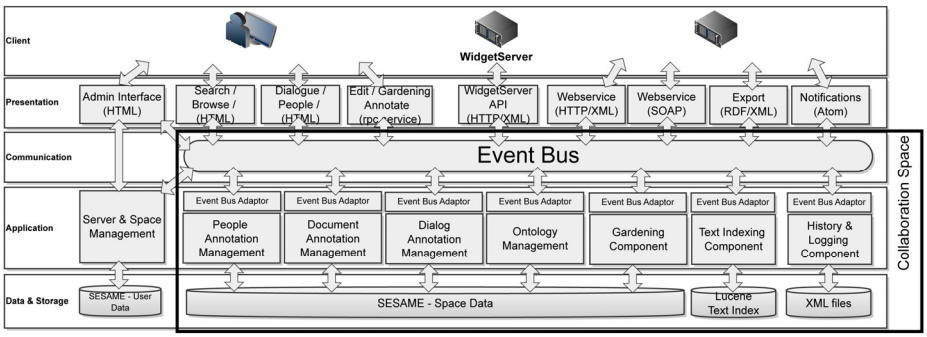


Fig. 1. Overview of SOBOLEO's architecture

Technically each collaboration space is represented by an **Event Bus** component that manages the communication within the space. Different parts of the functionality

are realized as services (both local within the server and remote) that are registered to the event bus. These services have access to a shared SESAME triple store and to disk space to store files (e.g. for the text index and logging information). On the server each collaboration space is represented as one folder that can even be moved between SOBOLEO installations.

Communication within the space is organized around the concept of **Events**. Any change is represented as a **Command Event** object, a query as a **Query Event** and any notification as a **Notification Event**. The event bus routes these events (and results) between the requester (mostly in the presentation layer) and the services that can process it. For example, a delete-concept command event is processed in the following way: 1) the event bus asks all registered command processing services to extend this command event with implied commands. In this example implied commands include the removal of relations that start or end in the deleted concept. 2) Next a different class of services is asked whether this event is permitted – this tests the user credentials sent with the event as well as the adherence to SKOS integrity constraints. 3) Command processors actually execute the change. 4) All registered event listeners are notified of the changes that have been done.

All events exist as Java objects, as JavaScript objects (allowing to create and receive them within AJAX applications), as XML serializations (allowing to create and receive them by applications written in any language) and as SOAP methods (allowing to create and receive events them from any SOAP client). Thanks to these interfaces almost all conceivable remote clients can do arbitrary changes to the collaboration space. The interfaces also allow to poll for recent events – enabling remote clients to stay up to date with the changes to the information space. We also supply a (Java) client library that takes care of the communication details and manages a local copy of the ontology (kept up-to date by polling the server in customizable intervals). In addition to these custom interfaces, SOBOLEO supports export of space data as RDF/XML and notification about changes to the information space in the ATOM format.

SOBOLEO is implemented in Java 6 on top of the Apache Tomcat 6.0 application server (http://tomcat.apache.org). The AJAX interfaces of the ontology editor and annotation tool are created with Google's Web Toolkit framework (http://code.google.com/webtoolkit). For the storage of RDF data we use Sesame 2.3.0 (http://www.openrdf.org/) and the text index is built on Apache Lucene 2.9.1 (http://lucene.apache.org).

4 User Interface & Functionality

SOBOLEO supports five function groups (detailed below) to support a group of people in the joint structuring of an information space containing an ontology, data about documents and people. Following the ideas of Ontology Maturing, SOBOLEO supports both development and use processes – to seamlessly support the refinement of the ontology immediately whenever a deficiency becomes apparent in use.

- The **ontology editor** enables users to structure the concepts with hierarchical relations (broader and narrower) and to indicate that concepts are “related”. Concepts can have a (multi-word) preferred label and a description in multiple languages; they can have any number of alternative and hidden labels. The collaborative editor can be used by several users at the same time. Changes are immediately visible and effective to all users and the ontology's usage (for search and in the web interface).

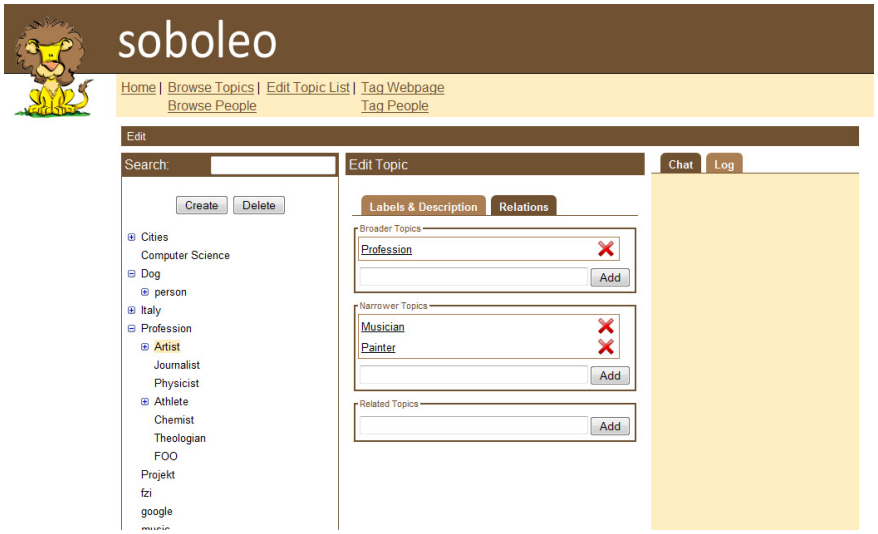


Fig. 2. User interface of the ontology editor

- Users can add semantically annotated web-documents to the shared information space through a **document annotation interface** that is available both as a bookmarklet and a web page. For annotating the resource, users can use any concept from the ontology or arbitrary (multi-word) tags. New tags are automatically added to the ontology as "prototypical concepts"; users can later consolidate and move them within the ontology. When saving the bookmark and annotation, the SOBOLEO system automatically sends out a crawler to fetch and index the contents of the page. The crawler is able to parse HTML, pdf and both the 2004 and 2007 MS Office formats.
- **Annotating people** works similarly to annotating web-documents. The primary idea is to annotate a person (identified by his/her email address) via his/her personal webpage, e.g. in the intranet, on the company's website or on a social network site. Each person that is tagged at least once is represented by one page within SOBOLEO and can also be tagged directly on this page.
- SOBOLEO’s **semantic search engine** enables users to search and retrieve annotated documents and people. The users can type their search terms into a text field—similar to common internet search engines. The semantic search

engine analyzes the entered search string for occurrences of concepts from the ontology. If it recognizes references to concepts, it searches for documents and people annotated with these concepts or narrower ones. At the same time it also searches the full text of all annotated webdocuments. On the result page the users get feedback on which concepts it understood the query to be referencing. Depending on the search string, the results and the ontology, the system may also propose a number of query refinements or relaxations.

- The **browsing interface** enables users to navigate through the ontology and the directory of annotated documents and people.

Note that the service oriented structure of the SOBOLEO system and the multitude of remote interfaces allow easily adding more functions to the SOBOLEO system. For example in the past we have extended it with capabilities to support structured dialogs about proposed ontology changes.

5 Related Work and Conclusions

There is a large number of applications that support groups of users in the collaborative creation of semantic data – the most prominent ones being the semantic wikis and particularly SMW [3]. SOBOLEO differs from these approaches in the use of a more specialized user interface geared towards the organization of data external to the system. PoolParty (http://poolparty.punkt.at/) also offers a web-based SKOS editor, but lacks comparable tools for the management of relevant web pages and people. Finally there is also a large number of social semantic bookmarking systems that combine social bookmarking with some semantics (see [4] for an overview), however, none of these has a similar functionality with respect to supporting multiple collaboration spaces or people tagging.

Acknowledgements. This work was supported by the MATURE EU IP under contract no. 216356.

6 References

1. Braun, S., Schmidt, A., Walter, A., Zacharias, V.: Using the Ontology Maturing Process Model for Searching, Managing and Retrieving Resources with Semantic Technologies. In: OTM 2008, LNCS vol. 5332, pp. 1568-1578, Springer, Heidelberg (2008)
2. Zacharias, V., Braun, S.: Tackling the Curse of Prepayment - Collaborative Knowledge Formalization Beyond Lightweight. In: 1st Workshop on Incentives for the Semantic Web, ISWC2008, Karlsruhe, Germany (2008)
3. Krötzsch, M., Vrandecic, D., Völkel, M., Haller, H., Studer, R.: Semantic Wikipedia. In: Journal of Web Semantics vol. 5, pp. 251–261, Elsevier (2007)
4. Braun, S., Schora, C., Zacharias, V.: Semantics to the Bookmarks: A Review of Social Semantic Bookmarking Systems. In: I-SEMANTICS 2009, pp. 445-454, Verlag der Technischen Universität Graz

Collaborative Metadata Editor Integrated with Ontology Services and Faceted Portals

Jussi Kurki and Eero Hyvönen

Semantic Computing Research Group (SeCo)
Aalto University, School of Science and Technology, and University of Helsinki
http://www.seco.tkk.fi/, firstname.lastname@tkk.fi

Abstract. This paper presents a generic RDF metadata editor SAHA 3 for collaborative content creation and instant semantic content publishing on the Semantic Web. SAHA 3 is a combination of a user-friendly interface and rich editing tools, that are able to utilize external ONKI ontology repositories as services. The system is integrated with a faceted portal engine HAKO, by which the metadata can be published instantly as a semantic faceted portal. Using the SAHA-ONKI-HAKO integrated system, a semantic portal can be created very easily by end-users by defining metadata schemas and related vocabularies, by annotating content, and by interactively configuring the user interfaces of the editor and the search engine. The system is in use in several semantic web applications and scales up to hundreds of thousands content objects.

1 Introduction

The basic process of implementing a faceted (semantic) portal [1–3] includes the following major steps: 1) Formulate vocabularies/ontologies/facets for representing domain concepts. 2) Design metadata schemas for representing content using (1). 3) Annotate content using (1) and (2), typically by a group of distributed peers in a Web 2.0 fashion. 4) Select the facets (1) and create the portal with semantic search and browsing facilities. This paper presents the tool SAHA 3[1] for the latter two phases. The idea is that given a set of domain vocabularies/ontologies/facets and a metadata schema for annotation (phases 1 and 2), a web-based annotation facility for distributed semantic content creation can be created instantly without programming skills. In a similar way, a faceted portal is automatically and instantly created online after annotating the content by just selecting facets for searching. Again, programming is not needed but only configuring the system using a web-based interactive interface.

We first describe the main features of the SAHA 3 metadata editor for the content creation phase, and then the integrated faceted portal engine HAKO[2] for content publishing, followed by notes about implementation and scalability. Finally, contributions of the work w.r.t related systems are discussed, and some application use cases are listed.

[1] http://www.seco.tkk.fi/services/saha/
[2] http://www.seco.tkk.fi/tools/hako/

2 SAHA 3 Metadata Editor Features

The original requirements for the web-based annotation editor SAHA [4] are simplicity (hiding technical concepts related to markup languages and ontologies from its user), adaptivity (to different metadata models), quality (helping and guiding the annotator to good and correct annotations), collaboration (supporting distributed simultaneous annotation at different locations), and portability (using the system on the web without installing any special software). SAHA 3 is a completely re-written version of SAHA with the following main new features: First, there is more support for general RDF editing, e.g. for inline editing of nested metadata, and for extending internal vocabularies. Second, the system is scalable to large datasets up to hundreds of thousands of objects. Third, SAHA 3 incorporates a simple publishing platform for building end-user search portals with full-text and multi-faceted search. The whole pipeline[3] from metadata editing to the end-user portal application is accessible and configurable through a web-based interface.

During annotation, references to external ontologies are handled using the ONKI web service interface [5, 6]. SAHA 3 utilizes autocompletion [7, 8] as a key component to find references. When the user tries to find a concept, SAHA 3 uses at the same time web services to fetch concepts from connected external ONKI ontology repositories, and a local index to find locally defined concepts. Results are shown in one autocompletion result list regardless of origin. The same query can cover several ontology repositories at the same time. References to resources within the project at hand and external to it (in an external ontology repository) are transparent to the user.

The inline editor is new feature in SAHA 3 that has been found very handy by end-users. The idea is simple: a resource referenced through an object property can be edited inline at the right location in a small version of the editor inside the existing editor. In this way, several levels of editors can be opened recursively within each other, and the RDF network can be edited without moving from one resource window to another. For example, in Figure 1 the FOAF profile of the first author of this paper is edited, and the value of the property "knows" is opened inline as a similar profile editor for the second author of this paper. Although the user interface easily becomes cluttered after a few levels, the inline editor is a handy way to add and edit nested metadata, as in this example shows.

SAHA 3 supports collaborative simultaneous editing. Resources that are being edited by one user are locked from other users. A chat facility has been implemented in the editor to facilitate instant discussions between peer editors (cf. the upper right corner in Fig. 1).

3 Faceted Search Engine HAKO

A SAHA 3 project can be published through the search interface of the HAKO portal engine. HAKO supports both free-text and faceted search [1–3]. For exam-

[3] SAHA Sandbox: http://demo.seco.tkk.fi/saha3sandbox/saha3/main.shtml

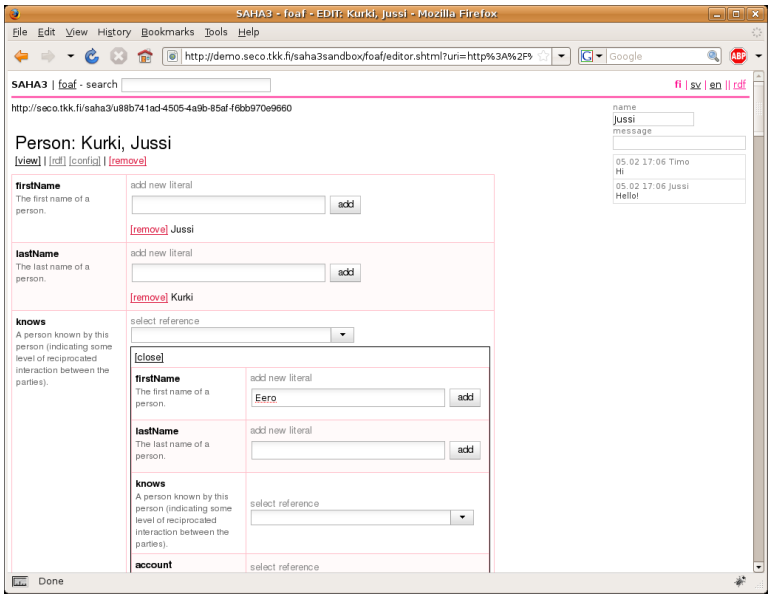


Fig. 1. Editing FOAF profiles in the SAHA 3 editor. New value for "knows" is created inline as a new instance of a person.

ple, Fig. 2 depicts a HAKO application using the RDF store of the Kirjasampo-system[4] containing tens of thousands of instances of literary work of different kinds. The user has entered the keyword "tolstoi". Free-text search is done as a prefix search by default. On the left, one can see the hit list of 70 books and plays related to Tolstoi distributed over the facet categories, from where the user can refine the search either by type or by theme. The facets are configurable—the administrator can select any object property to be a facet top category.

Apart from the facet administration, HAKO interface is actually only a front-end to the SAHA 3 model. The configurable, shared data model (including the indices), and the dataset used by SAHA 3 are the same as those used by HAKO. This means that all modifications made with the SAHA 3 editor are reflected on the HAKO interface instantly.

The system is implemented in Java on top of Spring[5] framework. The data model is based on TDB[6] RDF database. Full-text search is backed by Lucene[7]. The editor interface is built using DWR[8] and Dojo[9] AJAX-components.

[4] http://kirjasampo.fi/
[5] http://www.springsource.com/
[6] http://openjena.org/TDB/
[7] http://lucene.apache.org/
[8] http://directwebremoting.org/
[9] http://www.dojotoolkit.org/

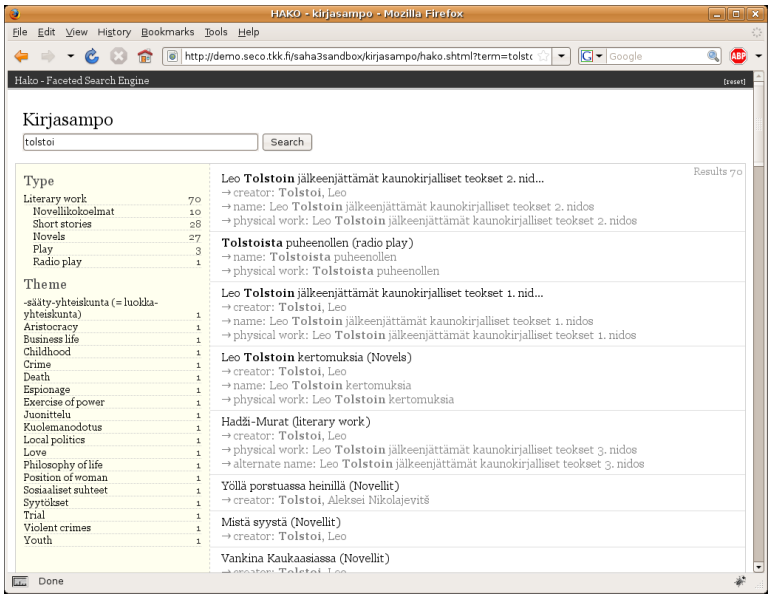


Fig. 2. SAHA3/HAKO multifaceted search interface on "Kirjasampo"-project.

4 Discussion

SAHA 3 makes use of external distributed ONKI ontology repository services, which is very handy when utilizing external third party ontologies and large vocabularies. From the interface design viewpoint, inline editing has been found a very useful feature. A major novelty of the system is the integration of the editor with a portal engine in real time. The idea of instant creation of a faceted search engine bears resemblance with SIMILE Exhibit[10]. However, in our case the system is integrated with a metadata editor, ontology services, and Semantic Web data models, and the search technology is based on Lucene (and not JavaScript) scaling up to very large datasets. SAHA 3 has been used in projects containing over 100,000 instances, and it has not shown signs of slowing down, or requiring large amounts of memory. The Tomcat server[11] used runs well below 500MB with large projects.

SAHA 3 is in production use in the HealthFinland[12] system and the Kirjasampo project, where some 50 librarians have been annotating metadata about tens of thousands of novels, short stories, authors, and other objects related to Finnish literature for a semantic portal. The editor has been used is many ways

[10] http://www.simile-widgets.org/exhibit/
[11] http://tomcat.apache.org/
[12] http://www.seco.tkk.fi/applications/tervesuomi/

in CultureSampo[13], e.g. for creating the semantic narrative descriptions of the Finnish History Ontology[14] and the Semantic Kalevala epic[15]. HAKO has been used in several industrial application demonstrations for searching documents.

Acknowledgements This work is part of the National Semantic Web Ontology project in Finland[16] (FinnONTO, 2003–2012), funded mainly by the National Technology and Innovation Agency (Tekes) and a consortium of 38 organizations.

References

1. Pollitt, A.S.: The key role of classification and indexing in view-based searching. Technical report, University of Huddersfield, UK (1998) http://www.ifla.org/IV/ifla63/63polst.pdf.
2. Hearst, M., Elliott, A., English, J., Sinha, R., Swearingen, K., Lee, K.P.: Finding the flow in web site search. CACM **45**(9) (2002) 42–49
3. Hyvönen, E., Mäkelä, E., Salminen, M., Valo, A., Viljanen, K., Saarela, S., Junnila, M., Kettula, S.: MuseumFinland—Finnish museums on the semantic web. Journal of Web Semantics **3**(2) (2005)
4. Valkeapää, O., Alm, O., Hyvönen, E.: A framework for ontology-based adaptable content creation on the semantic web. Journal of Universal Computer Science **13**(12) (2007)
5. Viljanen, K., Tuominen, J., Hyvönen, E.: Ontology libraries for production use: The Finnish ontology library service ONKI. In: Proceedings of the ESWC 2009, Heraklion, Greece, Springer–Verlag (2009)
6. Tuominen, J., Frosterus, M., Viljanen, K., Hyvönen, E.: ONKI SKOS server for publishing and utilizing SKOS vocabularies and ontologies as services. In: Proceedings of the ESWC 2009, Heraklion, Greece, Springer–Verlag (2009)
7. Hyvönen, E., Mäkelä, E.: Semantic autocompletion. In: Proceedings of the First Asia Semantic Web Conference (ASWC 2006), Beijing, Springer–Verlag (2006)
8. Hildebrand, M., van Ossenbruggen, J., Amin, A., Aroyo, L., Wielemaker, J., Hardman, L.: The design space of a configurable autocompletion component. Technical Report INS-E0708, Centrum voor Wiskunde en Informatica, Amsterdam (2007)

[13] http://www.seco.tkk.fi/applications/kulttuurisampo/
[14] http://www.seco.tkk.fi/ontologies/histo/
[15] http://www.seco.tkk.fi/applications/kulttuurisampo/kalevala/
[16] http://www.seco.tkk.fi/projects/finnonto/

Linked Open Ontology Services

Kim Viljanen, Jouni Tuominen, Mikko Salonoja and Eero Hyvönen

Semantic Computing Research Group (SeCo)
Aalto University, School of Science and Technology, and University of Helsinki
http://www.seco.tkk.fi/, firstname.lastname@tkk.fi

Abstract. Ontology repository systems are used for publishing and sharing ontologies and vocabularies for content indexing, information retrieval, content integration, and other purposes. However, interlinking these distributed repositories to provide global search and browsing over the repositories has not been made. In the spirit of Linked Open Data, we propose creating a network of Linked Open Ontology Services (LOOS) consisting of ontology repositories that publish their content using a shared API. To test the approach, we have defined an HTTP API and present a proof-of-concept implementation consisting of three client applications that are used for accessing a LOOS network of over 50 ontology servers, part of the Ontology Library Service ONKI.

1 Introduction

Ontology repositories have been considered a key resource for building a global infrastructure to enable the vision of the Semantic Web [1]. Many ontology repository systems exist for publishing and sharing ontologies and vocabularies for content indexing, information retrieval, content integration, and other purposes, e.g. Cupboard [2], BioPortal [3], OOR [4], and ONKI [5].

However, currently each ontology repository is a separate island, with no connections with other repositories. This means that, e.g. global search, browsing, or inference over the repositories can not be done, which creates a hindrance for using the ontologies globally. For example, searching for the concepts with the label "fish" from all existing ontology repositories around the Internet is currently not possible although many ontology repositories surely contain ontologies with matching concepts. Because the user may not find the correct ontology or concept for one's needs, it means that:

- the quality of annotations may decrease if the optimal concept is not found,
- redundant new ontologies are created if the existing ontologies are not found,
- interlinking of data decreases due to creating redundant ontologies, and
- merging data for semantic web applications becomes more difficult due to the need for ontology matching.

As a solution to the problem of how to access the repositories globally, in the spirit of the Linked Open Data[1] [6], we propose a Linked Open Ontology

[1] http://linkeddata.org

Services (LOOS) architecture consisting of ontology repositories that publish their content through common API, and thus making it possible to access the different ontology services in a distributed way.

In the following, we present the proposed architecture and API. Then, a proof of concept implementation of clients and servers is described. Finally, related work is discussed and contributions of the paper summarized.

2 LOOS API and Metadata About Services

The LOOS Network consists of ontology repositories that publish their content using a common, uniform LOOS API. A key idea of the LOOS API is to hide the ontology schema specific representations (such as OWL and RDFS) behind a uniform, simplified, SKOS-like representation of the key elements of the ontology: the concepts and their relations. Hiding ontological details makes it easier both to provide a uniform API to the ontologies and to display the search results from the underlying repositories in a uniform way to the user. After finding the matching ontology or concept, if the full power of a specific ontology representation is needed, the user can be directed to the specific ontology repository.

The main methods of the API are following:[2]

- *search*: search concepts
- *getLabels*: get the labels of a concept
- *getEquivalentConcepts*: get the equivalents of a concept
- *getConceptHierarchy*: get the concept hierarchy of a concept
- *getFullPresentation*: get all the information about a concept
- *getDirectory*: get the directory view of an ontology

The search method is used for finding concepts using various restrictions, such as text, concept type, or parent concept. The method returns a list of matching concepts. To get information about each concept, the properties and relations of a concept can be queried using methods such as *getLabels*. For efficiency, the *getFullPresentation* method returns all information about a certain concept in a single request. When browsing an ontology, an overview of the ontology is helpful. To support this, the *getDirectory* method returns an ontology-specific overview of the given ontology, consisting of e.g. the alphabetical ordering of the concept labels, manually defined grouping of concepts, or the top concepts of the ontology.

We propose implementing the LOOS API as a lightweight, stateless, and cacheable HTTP GET based API that returns data using the JSON format which provides an easy way to add LOOS support to ontology servers.

To find LOOS enabled services, metadata about the services is needed. We use the property *loos:APIBaseURL* for describing the URL of each LOOS endpoint. Additional information about the ontology such as title and description may be expressed using e.g. the Dublin Core metadata schema. Based on the metadata, e.g. a directory of LOOS services can be published.

[2] For the complete API documentation, check: http://www.yso.fi/loos/

3 The Proof-of-Concept Implementation

The ONKI SKOS ontology server has been used for publishing over 70 ontologies in the Finnish Ontology Library Service ONKI [7]. Global search to the ontologies was however not possible, because the ontologies were running as separate server instances. To solve this problem, and as a proof-of-concept implementation for the LOOS aproach, the LOOS API was implemented to the ONKI SKOS server and the server instances were described with LOOS metadata (see Fig. 1).

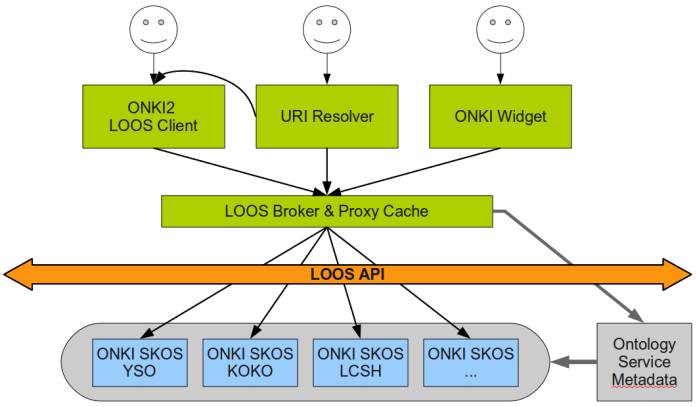


Figure 1. The proof-of-concept implementation of LOOS.

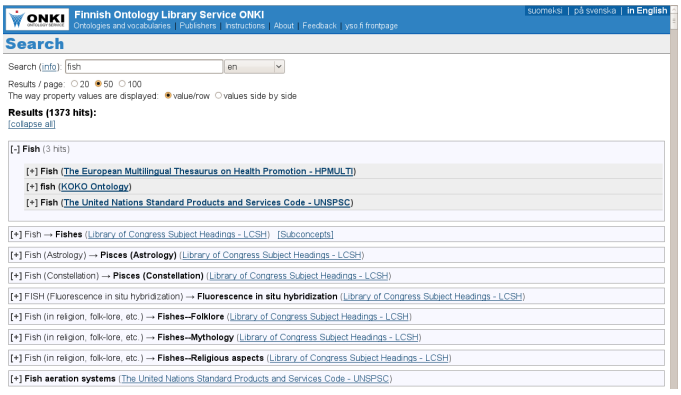


Fig. 2. The global LOOS search for "fish" matches many ontologies.

The ONKI2 Browser[3] is a global search and browsing user interface for accessing the LOOS network of ontology servers (the ONKI SKOS servers) in a uniform way . For example, making a global query to all ontology servers can be done (see Fig. 2). ONKI2 was mostly implemented using PHP[4].

Another client is the JavaScript-based ONKI Selector widget [8] for adding ontological concept search to HTML forms which now supports using the LOOS network as a back-end. As a third client, we implemented also a URI resolver for dereferencing the end-user's ontology concept URI requests to a suitable representation provided via the LOOS network, such as HTML or RDF.

To make client implementation easier, a broker for accessing the LOOS network was implemented. It provides a registry of LOOS enabled ontology servers based on the ontology service metadata, a single access point for using the LOOS network and a cache[5] to speed up potentially slow HTTP requests to individual ontology servers. Based on the registry, the broker directs LOOS requests to relevant back-end ontology servers.

4 Discussion

Compared to more general methods of accessing RDF data, such as SPARQL[6] and Linked Data [6], the LOOS approach focuses on ontologies. For example, when querying for the concept hierarchy of a concept via the LOOS API, one does not need to know what RDF properties (e.g. *rdfs:subClassOf* or *skos:broader*) are used in the data to express the hierarchical relations between concepts. The LOOS API restricts the set of possible queries, which makes it easier to implement and use the API in the ontology servers and the client applications.

APIs for accessing ontologies and vocabularies presented previously include the SKOS API[7] and the OWL API[8]. Compared to them, the LOOS API provides a higher abstraction, independent from specific ontology languages. Compared to the API's of BioPortal [3], Swoogle[9] and Watson[10], the goal of LOOS is to create a network of ontology servers based on a shared API that is implemented by all services. Therefore, the LOOS API focuses on a few basic methods that reflects the basic functionality of ontology repositories, e.g. concept search. In future, the LOOS API should be made compatible with repository specific APIs.

Ontology servers such as BioPortal and Cupboard support publishing interlinked ontologies, but the ontologies have to be uploaded into a centralized service for a global search. In contrast, in the LOOS approach ontologies can be published using a ontology service that is optimized for the specific ontology and

[3] http://www.yso.fi/onki2/?l=en
[4] http://www.php.net/
[5] As a proxy cache we used Varnish: http://varnish-cache.org/
[6] http://www.w3.org/TR/rdf-sparql-query/
[7] http://www.w3.org/2001/sw/Europe/reports/thes/skosapi.html
[8] http://owlapi.sourceforge.net/
[9] http://swoogle.umbc.edu/
[10] http://watson.kmi.open.ac.uk/

the user's needs while publishing the ontology service's basic functionality via the LOOS API to connect the ontology service to a global network of ontology services.

The loosely coupled LOOS architecture has turned out to be a flexible solution which makes it easy to implement additional clients when needed. Making multiple HTTP requests to back-end servers may be slow, but in our test implementation this lag has not been a problem thanks to the proxy cache between the clients and the back-end servers.

To conclude, this paper argues that the various ontology servers on the web should be made accessible using a common API that would provide a simple but universal methods for accessing the ontology content. As a solution, we propose the LOOS API and a metadata schema for describing the services.

Acknowledgements This work is part of the National Semantic Web Ontology project in Finland[11] (FinnONTO, 2003-2012), funded mainly by the National Technology and Innovation Agency (Tekes) and a consortium of 38 organizations.

References

1. Hyvönen, E., Viljanen, K., Tuominen, J., Seppälä, K.: Building a national semantic web ontology and ontology service infrastructure—the FinnONTO approach. In: Proceedings of the ESWC 2008, Tenerife, Spain, Springer–Verlag (2008)
2. d'Aquin, M., Lewen, H.: Cupboard - a place to expose your ontologies to applications and the community. In: Proceedings of the ESWC 2009, Heraklion, Greece, Springer–Verlag (June 2009) 913–918
3. Noy, N.F., Shah, N.H., Whetzel, P.L., Dai, B., Dorf, M., Griffith, N., Jonquet, C., Rubin, D.L., Storey, M.A., Chute, C.G., Musen, M.A.: BioPortal: ontologies and integrated data resources at the click of a mouse. Nucleic Acids Research **37**(Web Server issue) (2009) 170–173
4. Baclawski, K., Schneider, T.: The open ontology repository initiative: Requirements and research challenges. In: Proceedings of Workshop on Collaborative Construction, Management and Linking of Structured Knowledge at the ISWC 2009, Washington DC., USA (October 2009)
5. Viljanen, K., Tuominen, J., Hyvönen, E.: Ontology libraries for production use: The Finnish ontology library service ONKI. In: Proceedings of the ESWC 2009, Heraklion, Greece, Springer–Verlag (2009)
6. Bizer, C., Cyganiak, R., Heath, T.: How to publish linked data on the web. http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/ (July 27 2007)
7. Tuominen, J., Frosterus, M., Viljanen, K., Hyvönen, E.: ONKI SKOS server for publishing and utilizing SKOS vocabularies and ontologies as services. In: Proceedings of the ESWC 2009, Heraklion, Greece, Springer–Verlag (2009)
8. Viljanen, K., Tuominen, J., Hyvönen, E.: Publishing and using ontologies as mashup services. In: Proceedings of the 4th Workshop on Scripting for the Semantic Web (SFSW2008), 5th European Semantic Web Conference 2008 (ESWC 2008). (June 1-5 2008)

[11] http://www.seco.tkk.fi/projects/finnonto/

A User Interface for Ontology Repositories

Jouni Tuominen, Mikko Salonoja, Kim Viljanen, and Eero Hyvönen

Semantic Computing Research Group (SeCo)
Aalto University, School of Science and Technology, and University of Helsinki
http://www.seco.tkk.fi/, firstname.lastname@tkk.fi

Abstract. Finding ontologies and concepts from a collection of ontologies is a recurring task in many use cases, such as content indexing, searching, and ontology developing. To facilitate this, efficient search and browsing methods are needed. This paper introduces ONKI2, an ontology browser providing a user interface for a repository of ontologies. The system provides a multi-facet search facility for finding an ontology. Finding concepts is supported by autocompletion-based text search that can be refined with additional restrictions. ONKI2 is in use in the Finnish Ontology Library Service ONKI for a collection of 79 ontologies and vocabularies.

1 Introduction

Tools for finding ontologies and concepts are needed in many use cases, such as ontology development, ontology-based content indexing, and searching [1]. When looking for a suitable ontology for a given task in an ontology repository [1–4], methods for filtering the ontologies based on the needs of the user are needed. This can be facilitated using the metadata of the ontologies (e.g. name, description, subject, type) and the contents of the ontologies (i.e. the concepts of the ontology). Also, getting an overview presentation of an ontology is important for understanding the domain, purpose, and structure of the ontology. When browsing an ontology, the concepts should be presented in a clear, meaningful way to the end-user.

This paper introduces ONKI2[1], an ontology browser providing a user interface for a repository of ontologies and vocabularies. The goal of ONKI2 is to provide means for finding and utilizing ontologies for content indexers, information searchers, and ontology developers. The introduced system is in use providing access to a collection of 79 ontologies of various domains. In the following, means for finding ontologies and concepts in ONKI2 are first presented. After this, the system implementation is described. In conclusion, related work is discussed and the contributions of the paper are summarized.

2 Finding Ontologies

Finding a suitable ontology in a repository of ontologies may be laborious without efficient filtering methods. To facilitate the search process of an ontology,

[1] http://www.yso.fi/onki2/?l=en

ONKI2 provides a multi-facet search interface integrated with text search. The idea is illustrated in Fig. 1. The available facets are the subject, structure, and publishing status of the ontology. The ontologies are assigned to the facets by utilizing the ontology metadata provided by the ontology developers, allowing filtering ontologies from different perspectives. The autocompletion text search is matched to the names and descriptions of the ontologies. In Fig. 1 the user is searching for a public (publishing status facet) advanced vocabulary (structure facet) in the domain of health or nature (subject facet) with a query string "mes".

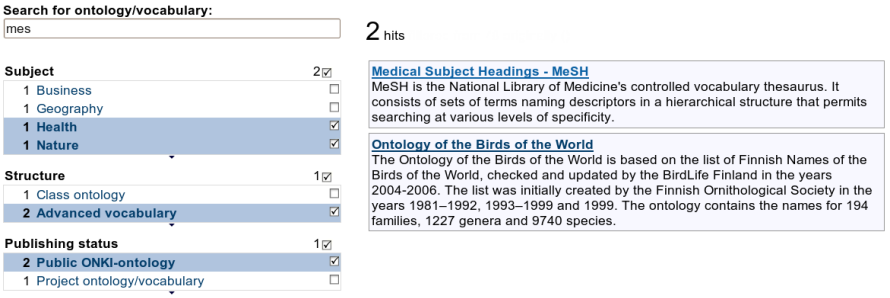


Fig. 1. Multi-facet search view of ontologies.

Once an ontology is found, it can be further examined by accessing the directory view of the ontology. The purpose of the directory is to give an overview of an ontology and act as a starting point for browsing it. Depending on the type of the ontology, the directory is either based on the groups categorizing the concepts, on the topmost concepts of the ontology (in SKOS vocabularies), or on the alphabetical ordering of the concepts. ONKI2 also supports searching ontologies based on their contents. An autocompletion text search is provided, matching to the labels of the concepts in all the ontologies of the collection. Thus, the user can find all the ontologies containing e.g. the concept "cat". Concept searching is discussed in more details in the following section.

3 Finding Concepts

For finding concepts in a collection of ontologies, ONKI2 provides a user interface with searching and browsing functionalities, as depicted in Fig. 2. Concepts can be searched by using autocompletion text search with a possibility to set additional restrictions. All the ontologies in the collection can be searched simultaneously, or alternatively the search can be limited to a single ontology. In both cases, the search is matched to the labels of the concepts. The search can be restricted to the concept labels in a specific language, to a specific concept type, to the group of concepts, or to the subconcepts of a specific concept.

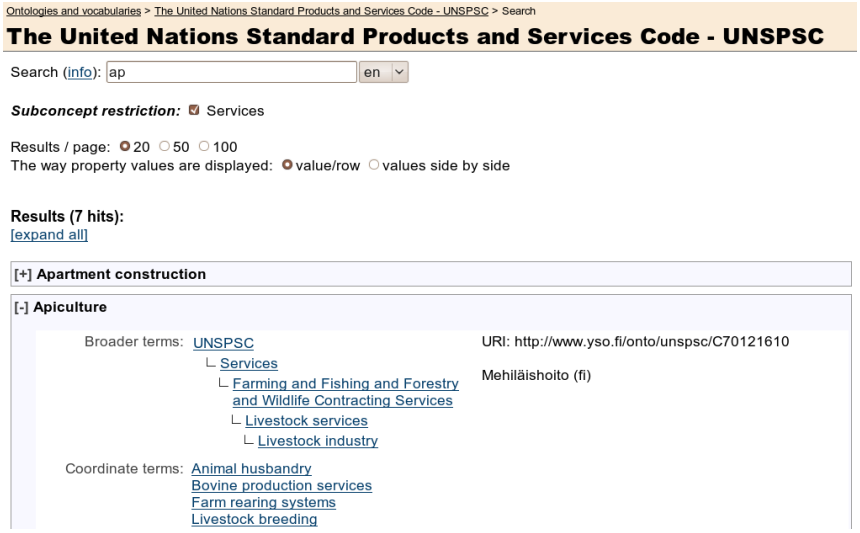


Fig. 2. Concept searching and browsing in the UNSPSC vocabulary.

The search result is presented as a list of concept label links. If there are several concepts with a same label, the concepts are grouped together assuming that they might be equivalent. When the user selects a concept, the properties of the concept are shown below the concept label, eliminating the need for unnecessary navigation between different views of the concepts. The properties are displayed in an ontology-specific meaningful order for the end-user in order to support concept understanding. The order can be specified by the developers of the ontology. Also, possible unnecessary properties can be filtered out from the property view. Moreover, the user can choose how the property values are displayed in the view. The values can either be presented compactly to minimize the used screen space, or more loosely for improved clarity. In the loose presentation mode, the superconcepts of the concept are presented as an indented hierarchy tree. In ontologies containing geographical information, a map user interface component can be used for visualizing the concepts, e.g. by placing geographical regions and points on the map.

All the property values of concepts that are resources (concepts, types, groups) are displayed as links leading to a corresponding concept or concept set, enabling the browsing of ontologies. This means that when selecting a concept from the concept hierarchy or among the related concepts of a concept, the properties of the selected concept are shown. If the selected resource is a concept group, the concepts belonging to the group are displayed as a search result list. Similarly, if a concept type is selected, concepts of that type are displayed. Mapping relations between ontologies can be used for inter-ontology browsing.

In Fig. 2 the user is searching for concepts in the vocabulary United Nations Standard Products and Services Code (UNSPSC)[2] that are subconcepts of the concept "Services". The string "ap" was used for querying the concept labels. From the search results, the user has selected the concept "Apiculture" for further investigation.

4 System Implementation

ONKI2 is implemented as a PHP[3] application by using the Zend Framework[4] for providing the Model-View-Controller architecture. The ontology multi-facet search view is generated with the SIMILE Exhibit Publishing Framework[5]. For processing the metadata of the ontologies, the ARC RDF library[6] is used. The map user interface component is provided by Google Maps[7].

The ontologies accessed with ONKI2 are represented by using RDFS, OWL, and SKOS. The ontologies are published in ONKI SKOS ontology server [5] instances. The ONKI SKOS servers are accessed by utilizing their HTTP API[8] for generating the ontology directory views, providing concept search and showing the properties of the concepts.

5 Discussion

The previously developed Ontology Library Service ONKI [1] supports publishing a collection of ontologies in different ontology servers, such as ONKI SKOS [5]. However, the ontologies are displayed only as a simple listing, and searching for concepts from all the ontologies simultaneously is not supported. Browsing between ontologies is possible if the ontologies contain mapping relations. BioPortal [2] is an ontology repository for accessing and sharing ontologies in the biomedical domain. The system is also used in the Open Ontology Repository Initiative [4]. It supports searching for concepts from all the ontologies and browsing between ontologies. Ontologies can be filtered based on their categories and groups.

Another ontology repository supporting concept searching and browsing of a collection of ontologies is Cupboard [3]. It provides an overview for an ontology by generating a graph of the key concepts of the ontology. It also supports collaboration by introducing commenting and reviewing options and shared spaces for publishing and grouping ontologies. Also semantic web search engines such as Watson [6], Swoogle [7], and Sindice[9] provide means for locating ontologies

[2] http://www.unspsc.org
[3] http://www.php.net
[4] http://framework.zend.com
[5] http://www.simile-widgets.org/exhibit/
[6] http://arc.semsol.org
[7] http://maps.google.com
[8] http://www.yso.fi/onkirest/
[9] http://sindice.com

and concepts on the web. However, as general RDF search engines, their support for understanding and browsing ontologies is rather limited due to the lack of visualization methods suited for ontologies.

ONKI2 is a general-purpose ontology browser for a collection of ontologies, supporting finding ontologies and concepts from the collection. The ontologies and their concepts are displayed in a meaningful, intuitive way for facilitating their understanding and usage as ready to use ontology web services [1, 5]. Furthermore, ONKI2 provides access to a network of distributed ontology repositories conforming to the Linked Open Ontology Services (LOOS) architecture [8].

Acknowledgements This work is part of the National Semantic Web Ontology project in Finland[10] (FinnONTO, 2003–2012), funded mainly by the National Technology and Innovation Agency (Tekes) and a consortium of 38 organizations.

References

1. Viljanen, K., Tuominen, J., Hyvönen, E.: Ontology libraries for production use: The Finnish ontology library service ONKI. In: Proceedings of the European Semantic Web Conference ESWC 2009, Heraklion, Greece, Springer–Verlag (2009)
2. Noy, N.F., Shah, N.H., Whetzel, P.L., Dai, B., Dorf, M., Griffith, N., Jonquet, C., Rubin, D.L., Storey, M.A., Chute, C.G., Musen, M.A.: BioPortal: ontologies and integrated data resources at the click of a mouse. Nucleic Acids Research **37**(Web Server issue) (2009)
3. d'Aquin, M., Lewen, H.: Cupboard - a place to expose your ontologies to applications and the community. In: Proceedings of the European Semantic Web Conference ESWC 2009, Heraklion, Greece, Springer–Verlag (2009)
4. Baclawski, K., Schneider, T.: The open ontology repository initiative: Requirements and research challenges. In: Proceedings of Workshop on Collaborative Construction, Management and Linking of Structured Knowledge at the ISWC 2009, Washington DC., USA (2009)
5. Tuominen, J., Frosterus, M., Viljanen, K., Hyvönen, E.: ONKI SKOS server for publishing and utilizing SKOS vocabularies and ontologies as services. In: Proceedings of the European Semantic Web Conference ESWC 2009, Heraklion, Greece, Springer–Verlag (2009)
6. d'Aquin, M., Baldassarre, C., Gridinoc, L., Sabou, M., Angeletou, S., Motta, E.: Watson: Supporting next generation semantic web applications. In: Proceedings of IADIS International Conference on WWW/Internet, Vila Real, Portugal (2007)
7. Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R.S., Peng, Y., Reddivari, P., Doshi, V., Sachs, J.: Swoogle: a search and metadata engine for the semantic web. In: Proceedings of the ACM International Conference on Information and Knowledge Management, New York, NY, USA, ACM (2004)
8. Viljanen, K., Tuominen, J., Salonoja, M., Hyvönen, E.: Linked open ontology services. In: Proceedings of the Workshop on Ontology Repositories and Editors for the Semantic Web (ORES 2010), the Extended Semantic Web Conference ESWC 2010, Heraklion, Greece (2010)

[10] http://www.seco.tkk.fi/projects/finnonto/

OWLGrEd: a UML Style Graphical Editor for OWL

Jānis Bārzdiņš, Guntis Bārzdiņš, Kārlis Čerāns,
Renārs Liepiņš, Artūrs Sproģis

Institute of Mathematics and Computer Science, University of Latvia,
Raina blvd. 29, LV-1459, Riga, Latvia
Janis.Barzdins@lumii.lv, Guntis.Barzdins@lumii.lv, Karlis.Cerans@lumii.lv
Renars.Liepins@lumii.lv, Arturs.Sprogis@lumii.lv

Abstract. There have been many attempts to visualize OWL ontologies but none of them is considered completely satisfactory and this is still an open problem. We propose a UML style graphical editor for OWL which not only visualizes ontologies using extended UML class diagram notation but also provides ontology editing facilities unavailable in most of the other tools. Moreover, the editor contains additional features for graphical ontology exploration and development including interoperability with Protégé 4.

Keywords: OWL, graphical editor, visualization.

1 Introduction

Thousands of ontologies were developed in the past years and surely more will be developed in the future, therefore availability of efficient ontology development tools including graphical editors is essential. Ontologies are usually described in Web Ontology Language (OWL) which originally was defined as an extension to RDF graphs, therefore one of OWL canonical forms is a set of subject-predicate-object triples. This format is very uniform, which makes it easy to parse and store in computers, but it is completely unusable for humans. Humans tend to think in terms of higher abstraction levels like classes, instances and relations, but the current ontology visualization tools like IsaViz [1], OWLViz [2], GrOWL [3], Welkin [4], etc. visualize ontologies by showing every RDF triple as two nodes with a labeled arrow between them. Thus the information gets cluttered and spread over a large area, making the structure hard to perceive.

For a graphical form to be useful it has to group related concepts together – the approach that has been successfully used, e.g. in UML class diagrams. Many concepts of OWL are very similar to those of UML class diagrams and therefore there have been attempts to define a UML profile for OWL [5] that would make it possible to use existing UML tools to create and visualize ontologies. However, OWL has more features than UML class diagrams like class expressions, anonymous classes, etc., which are commonly used, but have unintuitive graphical representation. Therefore, even though a UML profile is better than RDF graphs, it is still hardly comprehensible. Another option is to use Protégé OWL editor [6] that enables to load

and save ontologies, edit its classes and properties, and define a class taxonomy. It provides a detailed view for each concept in an ontology, but does not show well its overall conceptual structure.

Our proposal is to extend the UML class diagram notation with Manchester-like syntax for the missing OWL features (chapter 2), thus making the notation compact and comprehensible. We have developed an editor for this notation that has a number of features to ease ontology creation and exploration, e.g. different layout algorithms for automatic ontology visualization, search facilities, intelligent zooming, graphical refactoring and interoperability with Protégé (chapter 3). In fact, the application of UML class diagram notation to OWL is not completely new and has been implemented in TopBraid Composer [7]. However, it is based on simplified UML class diagram model, lacks graphical editing facilities, and the available graphical services are limited as well.

2 Compact OWL Graphical Notation

The proposed graphical notation is based on UML class diagrams. For most features there is one to one mapping from OWL to UML concepts, e.g. ontologies to packages, OWL classes to UML classes, data properties to class attributes, object properties to associations, individuals to objects, etc. Meanwhile there are added also a number of new graphical elements that are not part of UML notation. Classes have fields where OWL expressions can be inserted, e.g. equivalent class expressions, superclass expressions and disjoint class expressions. Similar fields are added to associations and attributes. Anonymous classes are shown as boxes with only equivalent class expression definition possibility. There are a number of ways to visualize anonymous superclasses:

1) as a textual expression inside a subclass box,

2) as a generalization line from the subclass to the corresponding anonymous class,

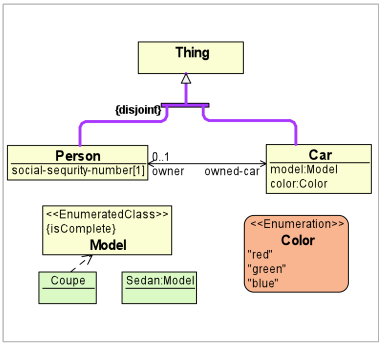
3) as multiplicity constraints in association, or

4) as a restriction line towards the corresponding class (e.g. the red line *eats* between *Lion* and *Herbivore* in fig. 2).

Ontology may be split into multiple diagrams inside a package with each diagram showing a different view of the ontology or its subset. Multiple generalization lines can be merged with a fork symbol, to reduce the number of incoming lines in a superclass. The editor provides means to switch from one graphical form to another (chapter 3), e.g. merge generalization lines within a fork.

To better explain the proposed notation let us consider an example in Figure 1. It shows a simple ontology representing people, cars, their properties and relations. This visualization uses only standard features of UML class diagrams. Classes are represented by rectangular boxes and data properties are shown as labels inside the class box. Object properties are represented with lines between boxes corresponding to their domain and range classes. If object property has an inverse then both are represented with the same line, e.g. *owner* and *owned-car*. Object properties can alternatively be shown as labels inside domain class box, e.g. object property *model*. Cardinality restrictions of such inline-shown properties are depicted in square

brackets next to the corresponding property name, in the same way as in regular class diagrams. If a range of a data property is an enumeration of values, then the enumeration is depicted as a box with rounded corners, e.g. *Color*. The fact that a class is defined by its instances is shown with a label <<*EnumeratedClass*>>, e.g. class *Model*.

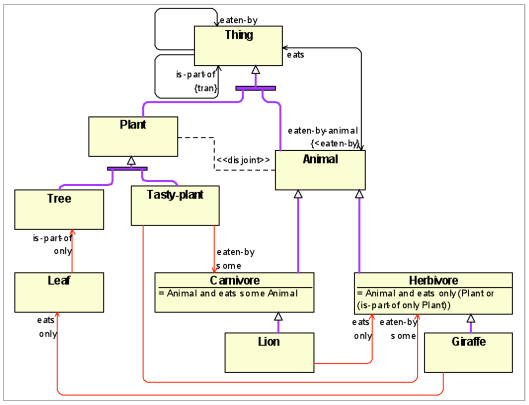



```
Class: owl:Thing
Class: Person
    SubClassOf:
        owl:Thing,
        social-sequrity-number exactly 1
    DisjointWith:
        Car
Class: Car
    SubClassOf:
        owl:Thing
    DisjointWith:
        Person
Class: Model
    EquivalentTo:
        {Coupe, Sedan}
Individual: Sedan
    Types:
        Model
Individual: Coupe
    Types:
        Model
DataProperty: color
    Domain:
        Car
    Range:
        {"blue" , "orange" , "red"}
```

```
ObjectProperty: property
    Domain:
        Person
    Range:
        Car
    InverseOf:
        owner
ObjectProperty: owner
    Characteristics:
        Functional
    Domain:
        Car
    Range:
        Person
    InverseOf:
        property
ObjectProperty: model
    Domain:
        Car
    Range:
        Model
DataProperty: social-sequrity-number
    Domain:
        Person
```

Fig. 1. Proposed graphical notation and corresponding Manchester notation

Next example already uses some graphical notations that are not a part of UML. Figure 2 depicts the popular African wildlife ontology. The red lines are 'some values from' and 'only values from' restrictions encoded as subclasses in OWL. It is obvious that this notation eases the comprehension of ontology, e.g. *Tasty-plant* must be eaten by some *Carnivore* and some *Herbivore*. Super properties are depicted as a text next to subproperty's name, e.g. *{<eaten-by}* next to subproperty name *eaten-by-animal* (symbol '<' corresponds to '*subproperty*' in UML notation).




```
Class: owl:Thing
Class: Leaf
   SubClassOf:
      is-part-of only Tree
Class: Plant
   SubClassOf:
      owl:Thing
   DisjointWith:
      Animal
Class: Tree
   SubClassOf:
      Plant
Class: Tasty-plant
   SubClassOf:
      Plant,
      eaten-by some Carnivore,
      eaten-by some Herbivore
Class: Animal
   SubClassOf:
      owl:Thing
   DisjointWith:
      Plant
Class: Herbivore
   EquivalentTo:
      Animal and
      (eats only
        (Plant or
          (is-part-of only Plant)))
```

```
Class: Giraffe
   SubClassOf:
      Herbivore,
      eats only Leaf
Class: Carnivore
   EquivalentTo:
      Animal
      and (eats some Animal)
Class: Lion
   SubClassOf:
      Carnivore,
      eats only Herbivore
ObjectProperty: eats
   Domain:
      Animal
   InverseOf:
      eaten-by-animal
ObjectProperty: is-part-of
   Characteristics:
      Transitive
ObjectProperty: eaten-by-animal
   Range:
      Animal
   InverseOf:
      eats
   SubPropertyOf:
      eaten-by
ObjectProperty: eaten-by
```

Fig. 2. Extended graphical notation and corresponding Manchester notation

3 Services of the Editor

A number of special services are implemented in our editor to ease ontology development. One of the services is graphical refactoring that allows modifying graphical notation without changing semantics as long as the same concept can be expressed through different constructions. This feature allows the user to choose the most compact graphical format depending on the context and the taste. One of the typical situations illustrating the need for graphical refactoring is generalization and fork: if there is a single super class with multiple incoming generalization lines, a fork can be added to reduce multiple lines into a single line, and vice versa.

When ontologies become large, their management becomes more difficult and additional features are required from the editor. First, a good automatic layout is crucial for understanding large ontologies and therefore several alternative layout modes are supported. Second, searching for the specific element in large ontologies may become painful and irritating without an appropriate service. A search mechanism implemented in our editor allows finding the necessary element by specifying the value for one of its text fields. For example, it allows finding classes by their name or the value of any other text field.

A more advanced service is full interoperability with Protégé 4, an editor widely used by ontology developers. The interoperability is implemented via custom Protégé plug-in that allows to send via TCP/IP socket an active ontology from our editor to Protégé, and vice versa. Ontologies in both directions are sent in interchange format, but generally any OWL serialization is acceptable. Interoperability allows ontology developers to use Protégé without changing their habits and afterwards visualize ontologies in external graphical editor using different automatic layout algorithms as well as further manual layout tuning. Moreover, a user can specify the way ontologies will be visualized by selecting notation options in preferences. In our graphical editor ontology developers can create new ontologies from scratch or alternatively graphically edit ontologies imported from Protégé; all graphically developed ontologies can afterwards be exported to Protégé from where they can be stored to various formats or checked with OWL reasoners.

4 Implementation

The editor is implemented using transformation driven architecture (TDA) [8, 9, 10] technology. TDA stores its information in the form of MDA-style models that are connected by model transformations. The user interface in TDA is implemented by means of universal engines (e.g. a graph diagramming engine, a property editor engine, etc.). Each individual tool (e.g. OWLGrEd) is created through a specially designed TDA tool definition configurator that creates instances of Tool Definition Model storing all meta information about an individual tool – element types, element styles, constraints and relationships among elements.

The Tool Definition Model instances are then interpreted by a universal interpreter that in cooperation with other TDA engines processes all end-user's actions.

Furthermore, for OWLGrEd, as for other tools, specific transformations can be created to support domain specific needs. In our case, only transformations supporting interoperability with Protégé, and specific attribute and annotation parsers had to be created. Thanks to the use of TDA it took only six person-months to produce the beta-version of the OWLGrEd editor, including development of Protégé plug-in and the design of the actual graphical notation (this has been the most time consuming part). The latest editor version can be downloaded from http://OWLGrEd.lumii.lv.

5 Conclusion and Future Work

In this paper we described a new, compact OWL graphical notation and a beta-version implementation of the actual graphical editor. Our notation is based on UML class diagrams with additional constructs for OWL specific concepts – our aim is to cover full OWL 2.0 specification. The editor has a number of features to ease ontology exploration and development, e.g. automatic layout algorithms and options for selecting which concepts shall be displayed. We are planning to add an option to store graphic layout information inside ontologies (we consider adding it as a special kind of annotations). We would also like to improve integration with Protégé, in particular, to synchronize ontologies in both tools after every editing step - current implementation allows exchanging only whole ontologies.

References

1. IsaViz, http://www.w3.org/2001/11/IsaViz/
2. OWLViz, http://www.co-ode.org/downloads/owlviz/
3. GrOWL, http://www.uvm.edu/~skrivov/growl/
4. Welkin, http://simile.mit.edu/welkin/
5. UML profile, http://www.omg.org/spec/ODM/1.0/PDF/
6. Protege, http://protege.stanford.edu/
7. TopBraid Composer, http://www.topquadrant.com/products/TB_Composer.html
8. Barzdins, J., Rencis, E., Kozlovics, S. The Transformation-Driven Architecture, Proc. of 8th OOPSLA Workshop on Domain-Specific Modeling. Nashville, USA, 2008, pp. 60-63.
9. Barzdins, J., Cerans, K., Kozlovics, S., Rencis, E., Zarins, A. A Graph Diagram Engine for the Transformation-Driven Architecture, Proc. of 4th International Workshop on Model Driven Development of Advanced User Interfaces (MDDAUI-2009). Florida, USA, 2009, pp. 29-32.
10. Barzdins, J., Zarins, A., Cerans, K., Kalnins, A., Rencis, E., Lace, L., Liepins, R., Sprogis, A.., GrTP: Transformation Based Graphical Tool Building, Proc. of 3th International Workshop on Model Driven Development of Advanced User Interfaces (MDDAUI-2007). Nashville, USA, CEUR Workshop Proceedings, http://ceur-ws.org, vol. 297.

OREMP: Ontology Reasoning Engine for Molecular Pathways

Renato Umeton[1], Beracah Yankama[1], Giuseppe Nicosia[2], and C. Forbes Dewey, Jr.[1]

[1] Massachusetts Institute of Technology, Cambridge MA 02139, USA,
oremp@mit.edu,
WWW home page: http://cytosolve.mit.edu
[2] University of Catania, Viale A. Doria 6, 95125 Catania, Italy

Abstract. The information about molecular processes is shared continuously in the form of runnable pathway collections, and biomedical ontologies provide a semantic context to the majority of those pathways. Recent advances in both fields pave the way for a scalable information integration based on aggregate knowledge repositories, but the lack of overall standard formats impedes this progress. Here we propose a strategy that integrates these resources by means of extended ontologies built on top of a common meta-format. Information sharing, integration and discovery are the primary features provided by the system; additionally, two current field applications of the system are reported.

1 Introduction

An increasing number of quantitative biomolecular pathway databases are updated and curated on a regular basis [1, 2], because molecular processes are being characterized and their descriptions shared continuously. Substantial effort has been devoted to the creation of searchable biological resources (such as GO [3] and UniProt [4]) which are publicly available, but there are semantic obstacles that inhibit their combined use. Different languages (*i.e.*, the data formats) are spoken by the data sources; there are different abstraction levels; and there is a lack of an overall frame capable of identifying overlaps and duplications [5]. Some syntactic conversions are available among pathway data-formats, and the state of the art for adjudication of the discrepancies between two SBML [6] models is semanticSBML [7], which exploits machine-readable information and the user input to create a merged SBML model. In the context of large-scale composite biological pathways, the merged-model approach is undesirable because it destroys the original component models and interrupts the curation process. For more than two SBML files, the tool must be run repeatedly with user-input, subjecting it to increasing human error, and suggesting that the order in which the models are aligned matters. An alternative approach based on the use of ontologies discerns when and on which topics models are a relevant part of the large-scale context. The state of the art is represented by BioPortal [8] which provides uniform access to most of the biomedical ontologies through a single

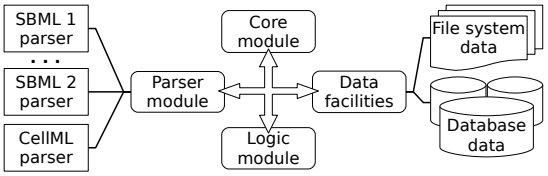


Fig. 1. System components are integrated to work together preserving a flexible and easily extensible architecture. Each module has different versions used on the basis of job in progress (*e.g.*, to parse an SBML file, will be dynamically chosen the SBML parser).

Entity	*has*
Annotation	type:STRING, uri:STRING, information:STRING.
Species	name:STRING, internalId:STRING, initialValue:REAL, inPathway:PATHWAY, hooks:SET_OF_ANNOTATIONS.
Kinetic reaction	internalId:STRING, kinetics:FORMULA, kineticParameters:SET_OF_PARAMETERS, inPathway:PATHWAY, reactants:SET_OF_SPECIES, catalysts:SET_OF_SPECIES, products:SET_OF_SPECIES, hooks:SET_OF_ANNOTATIONS.
Parameter	name:STRING, value:REAL.
Pathway	fullname:STRING, hooks:SET_OF_ANNOTATIONS.

Table 1. Main components of the minimalistic quantitative MIRIAM-compliant ontology used to abstract heterogeneous resources associated with biomolecular pathways. The format "attribute:REPRESENTATION" is used.

user-interface and advanced tools to query over biomedical data resources. Still, there are a lack of strategies for the database and ontology integration of quantitative biological sources written in different standards (*e.g.*, SBML and CellML [9]). What is described here is a system that creates extended ontologies out of different biochemical information sources and provides path duplication detection, sharing, integration, and knowledge discovery over heterogeneous resources. A prototype exists (MIT license, cf. http://cytosolve.mit.edu/oremp for software details) with utilities to export the extended ontologies in OWL format. This combination represents an Ontology Reasoning Engine for Molecular Pathways (OREMP). The OREMP framework creates extended ontologies out of different quantitative data formats and can be browsed at different levels of abstraction.

2 The Designed Framework

System Architecture. The system is composed of interchangeable and extensible components (Fig. 1). The four components interact as follows: (i) the *data access facilities* collect information about multiple pathways and existing biological databases; (ii) the *parser component* accesses different file formats (RDF, XML,

SBML, CellML, etc.) and extracts information from those sources; (iii) the *core module* assembles the knowledge from different sources into a coherent ontology (Table 1), and (iv) the *logic component* defines the conditions that identify when two biomolecular species are the same, or two reactions overlap. The combined execution of the two models without detecting reaction duplication will produce an incorrect evolution of species concentrations in time. This is a concrete, quantitative effect of incorrect ontology alignment. While the operational work-flow (i-iv) is kept fixed, it is of note that different versions of each component may be loaded by the system. A user-configurable algorithm chooses at run-time the components that are required for the current job. Whenever a new modeling standard is introduced, a new parser can be connected to OREMP to interface with it as well. Similarly, different users can define different versions of the *core component*, for example, according to their understanding about how the knowledge coming from different pathways should be aggregated. A useful analogy is the way modern graphics display programs seamlessly support different file formats (JPG, TIFF, DCM, etc.). Our approach is different from semanticSBML in that it provides the user the opportunity to exploit his/her understanding to define a consistent method of knowledge integration across ontologies. The independent curation process is preserved by maintaining the pathway identity, since the primitive element-pathway network is not destroyed by integration. Finally, we can optionally accept a dictionary of already aligned species, which can easily scale in the number of input pathways, as related in the next section.

Ontologies From Pathways. The system is constructed of three layers. The bottom layer represents the biochemical pathways, read in their primitive format (such as SBML and CellML). The second layer abstracts the pathways into a minimalistic and quantitative meta-format (sketched in Table 1) that includes all the MIRIAM [10] components. Annotations are preserved and extended with additional quantitative data to achieve a common description that can be represented as a single ontology. It is at this level that the extended ontology is primarily created. Entities and relations created in this manner are homogeneous in the ontological sense. This implies that several pathway collections can be combined in an ontology repository while maintaining a common semantic, meaning that the following steps can now be taken:
Sharing. Despite disparate initial data formats, the biochemical information described in each pathway is now homogeneously represented. This enables the direct reuse of componets (such as species or reactions) coming from different sources.
Integration. Our system ensures a consistent merging of the resources, automatically aligning the species and showing the end-user possible duplications among reactions in the different pathways.
Knowledge discovery. Once the species alignment is done and duplicate reaction have been detected, a new step is taken: for each reaction in each pathway the set of "alternative circuits" is computed. This means that given an arbitrary number of pathways, the system will identify all of the alternative ways to traverse from state S_0 to a state S_1 (where the states are different species config-

urations) within the overall set of reactions. In the last layer, all the information gathered is exported in OWL. With the OWL file we use the semantic tool, Protégé [11], to visually edit, compare, and finalize the biochemical information. With the OWL query interface, the user can now formulate "semantically-enabled" queries that were impractical when dealing with the previously heterogeneous, unaligned data repositories.

3 Usage Examples

OREMP in Combining Pathways for Parallel Solution. This system is embedded in the latest release of Cytosolve [12]. Its contribution to the integration of runnable pathways is the detection of duplicated reactions among different models. No matter the models chosen for simulation, once the species are aligned, the system identifies duplication problems in the reaction-models. From the user point of view this process is transparent: he/she receives a warning message that details the duplicated reactions and is prompted to confirm conflict elimination, and to resolve any differences in reaction kinetic rate constants.

OREMP in Querying Large, Independent Sources of Pathways. Our prototype was tested against the entire Biomodels.net curated collection [1] that contains about 240 molecular pathways. The result of the analysis was an overall view of the database and a list of about 500 groups of overlapping reactions. This analysis took 50 seconds on a single-core 2GHz Intel CPU. The previously described knowledge-discovery-step was taken on these resources as well. For each species configuration in the database, all alternative circuit paths were computed. This took about 2 hours on a quad-core 2GHz AMD CPU and resulted in a dictionary of thousands "biological equivalent" circuits (*i.e.*, equivalent reaction compositions). The latter experiment provides an interesting overview of the BioModels.net collection that we think can be used to boost the pathway modeling step - it provides a searchable dictionary of pathway building blocks. Perhaps more importantly, from the prospective of those who curate collections of biochemical pathways, this framework can be used to find inconsistencies and redundancies within their repository.

4 Conclusions

To our knowledge this is the first time that the information coming from different biological data sources are aggregated into a single quantitative ontology that can be queried at multiple levels. As detailed in previous sections, the OREMP application can combine several pathways, merge and combine pathway repositories, or revert to the original pathways, and inspect single-model details and query external repositories (such as UniProt and GO) referenced in pathway element annotations. Our system is independent of the different file formats in which the pathways are written and contains an extensible collection of parser modules. We have selected OWL as export format for the extended ontologies

and have adopted Protégé as our "Data Warehouse" for information storage, retrieval and reasoning. This framework transforms biomolecular pathways into extended ontologies to support knowledge sharing, integration and discovery. Since we generate the ontologies from a common semantics, the latter features are maintained when pathway collections are used to fill ontology repositories.

References

[1] Le Novère, N., Bornstein, B., Broicher, A., Courtot, M., Donizelli, M., Dharuri, H., Li, L., Sauro, H., Schilstra, M., Shapiro, B., Snoep, J., Hucka, M.: BioModels database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. Nucleic Acids Research **34**(Database issue) (2006) D689–691

[2] Lloyd, C.M., Lawson, J.R., Hunter, P.J., Nielsen, P.F.: The CellML model repository. Bioinformatics **24**(18) (2008) 2122–2123

[3] Ashburner, M., Ball, C., Blake, J., Botstein, D., Butler, H., Cherry, J., Davis, A., Dolinski, K., Dwight, S., Eppig, J., Harris, M., Hill, D., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J., Richardson, J., Ringwald, M., Rubin, G., Sherlock, G.: Gene ontology: tool for the unification of biology. the gene ontology consortium. Nature Genetics **25**(1) (2000) 25–29

[4] The UniProt Consortium: The universal protein resource (UniProt). Nucleic Acids Research **35**(Database issue) (2007) D193–197

[5] Bauer-Mehren, A., Furlong, L.I., Sanz, F.: Pathway databases and tools for their exploitation: benefits, current limitations and challenges. Molecular Systems Biology **5** (2009) 290–303

[6] Hucka, M., Finney, A., Sauro, H., Bolouri, H., Doyle, J., Kitano, H., and the rest of the SBML forum: The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics **19**(4) (2003) 524–531

[7] Krause, F., Uhlendorf, J., Lubitz, T., Schulz, M., Klipp, E., Liebermeister, W.: Annotation and merging of SBML models with semanticSBML. Bioinformatics (2009) btp642

[8] Noy, N.F., Shah, N.H., Whetzel, P.L., Dai, B., Dorf, M., Griffith, N., Jonquet, C., Rubin, D.L., Storey, M., Chute, C.G., Musen, M.A.: BioPortal: ontologies and integrated data resources at the click of a mouse. Nucleic Acids Research **37**(suppl_2) (2009) W170–173

[9] Lloyd, C., Halstead, M., Nielsen, P.: CellML: its future, present and past. Progress in Biophysics and Molecular Biology **85**(2-3) (2004) 433–450

[10] Le Novère, N., Finney, A., Hucka, M., Bhalla, U.S., Campagne, F., Collado-Vides, J., Crampin, E.J., Halstead, M., Klipp, E., Mendes, P., Nielsen, P., Sauro, H., Shapiro, B., Snoep, J.L., Spence, H.D., Wanner, B.L.: Minimum information requested in the annotation of biochemical models (MIRIAM). Nature Biotechnology **23**(12) (2005) 1509–1515

[11] Noy, N.F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R.W., Musen, M.A.: Creating semantic web contents with protege-2000. Intelligent Systems, IEEE [see also IEEE Intelligent Systems and Their Applications] **16**(2) (2001) 60–71

[12] Ayyadurai, S., Dewey, C.F.: Cytosolve: a scalable computational methodology for dynamic integration of multiple molecular pathway models. Cellular and Molecular Bioengineering (2010) In review

CONSISTOLOGY: A SEMANTIC TOOL TO SUPPORT ONTOLOGY EVOLUTION AND CONSISTENCY

Najla SASSI, Miracl Laboratory, sassinajla@yahoo.fr
Wassim JAZIRI, Miracl Laboratory,, wassim.jaziri@isimsf.rnu.tn
Faiez GARGOURI, Miracl Laboratory, faiez.gargouri@isimsf.rnu.tn

Abstract

Ontologies recently have become a topic of interest in computer science since they are seen as a semantic support to explicit and enrich data-models as well as to ensure interoperability of data. Moreover, supporting ontology's evolution becomes essential and extremely important, mainly when using ontologies in changing environments. An important aspect in the evolution process is to guarantee the consistency of the ontology when changes occur, considering the changes semantics. This paper proposes the Consistology tool developed to assist users in expressing evolution requirements and generating coherent ontology versions. This tool, based on coherent kits of change, has been experimented to evolve an ontology of education.

Key Words: Consistology, Semantic tool, Evolution kits, Ontology, Consistency.

1. Introduction

Changing environments require ontologies adaptable to changes that occur over time. The adaption of an ontology is a complex process and several evolution problems must be treated, in particular maintaining the ontology consistency after changing.

The application of a change on ontological entities is a modification of a subset of knowledge represented in the ontology. Change management requires defining mechanisms specifying how knowledge can be changed and how to maintain the consistency of knowledge after each change. In addition, ontological entities are semantically and conceptually linked, the application of a change in some ontological entities may have effects on other entities.

We are interested in this paper in defining evolution kits to allow updating ontologies while preserving their consistency. We also developed an ontology evolution tool 'consistology' to assist users in expressing evolution requirements and generating coherent ontology versions.

This paper is structured as follows. Section 2 presents an overview about the most representative approaches and tools used in ontology evolution. In Section 3, we propose our approach to support ontology evolution and to anticipate inconsistencies. Sections 4, 5 and 6 present the Consistology tool and its application to the education domain. Section 7 concludes this work.

2. State of the art

Several application areas are especially concerned with evolution of data and users requirements, such as software development [RL05], temporal databases [BB08] and ontologies.

Software systems are rarely stable following initial implementations. They have complex structures which are likely to continually undergo changes during their lifetime. Temporal databases support time-varying information and maintain the history of the modelled data. They allow the maintenance of data histories through the support of time semantics at system level. We refer to [BB08] [SBJ+10] for further information about related work on software development and temporal databases.

Ontologies, like software development and temporal databases, need to change every time the modelled real world has changed. Ontology evolution is the process of adaptation of ontology to evolution changes and the consistent management of these changes to guarantee the consistency of ontology when changes occur [KF01] [NK04]. It encompasses the set of activities, both technical and managerial, which ensures that ontology continues to meet organizational objectives and users needs in an efficient and effective way [Sto04]. According to [Sto04], "Ontology Evolution is the timely adaptation of ontology to the arisen changes and the consistent propagation of these changes to dependent artifacts." It concerns different aspects: the needs to update and to evaluate data, the changes to apply in conformity with these needs, the management of inconsistencies in all parts of the ontology as well as in the dependent artifacts.

According to [MS03], two types of inconsistency can be identified:

- Structural inconsistency occurs when the constraints of the ontology model are invalid or if the semantics of the subjacent language of ontology is not respected.
- Semantic inconsistency occurs when the significance of the entities of ontology is changed.

An ontology is considered consistent if its axioms are respected and if it satisfies the whole of the invariants defined in the model of ontology [MS03].

Stojanovic et al. [SSG+03] proposed an approach for the management of evolution and the maintaining of consistency for KAON ontologies. The authors proposed the concept of strategies of evolution which allow choosing the most suitable solutions for the resolution of inconsistencies

Haase et al. [HS05] also used the concept of strategies of resolution based on the constraints of OWL-Lite for the detection and the resolution of inconsistencies in OWL ontologies. However, the resolution of inconsistencies is done after application of changes. It is ensured in two phases: the detection of inconsistencies which consists in finding the parts of ontology which do not satisfy the consistency conditions and

the generation of changes that allow ensuring the consistency of ontology by generating additional changes.

Flouris et al. [FP05] differentiate between a consistent ontology and a coherent ontology. Ontology is inconsistent if there is no interpretation which satisfies all the axioms of this ontology. It is incoherent if it does not satisfy some predefined constraints or the related invariants. The predefined constraints describe the consistent model of ontology. These authors consider the inconsistencies as sign of bad design and their correction does not relate to the ontology evolution but it is rather related to the ontology design.

Luong et al. [LD07] distinguish two levels of consistency for the model of ontology: structural consistency and logical consistency. Structural consistency relates to the constraints of consistency defined for an ontology model by ensuring a good organization of the ontological entities at the level of structure. Logical consistency checks if the elements of ontology remained "semantically correct" after their evolution.

In [KJL09], the authors investigate how ontologies developed for use in Semantic Web technology could be used in checking the consistency of requirements specifications. They use reasoning which is a part of ontology. The *TESSI* tool has been developed.

Djedidi et al. [DA10] proposed an approach of enrichment of ontology with an aim of optimizing and automating the management of changes while ensuring the consistency and the quality of ontology after evolution.

The maintenance of consistency is ensured through alternatives of resolution of inconsistency. A model of quality is defined and applied to guide the resolution of inconsistencies and to evaluate the impact of the suggested alternatives on the quality. A prototype of the change management system was implemented to manage changes of OWL ontologies while maintaining their consistency and quality.

In addition, number of scientific and commercial tools for creating, managing and updating ontologies have been used to build applications in several domains such as KAON [OVM+04], OntoView [KFK+02], OntoManager [SSG+03], TextToOnto [MV01], SHOE [HH00], PromptDiff [NM02], Protégé[1], etc. Some tools dedicated to ontology debugging are also proposed, such as RADON [JHQ+09], SWOOP [KPS+05], DION [2], OntoClean [3], MUPSter [SC03] etc. Other tools, such as ConsVISor[4], do both consistency checking and debugging. A comprehensive survey on ontology editors and tools can be found in [Den09] [GM03].

[1] http://protege.stanford.edu/.
[2] http://wasp.cs.vu.nl/sekt/dion/.
[3] http://www.ontoclean.org/.
[4] http://projects.semwebcentral.org/projects/consvisor/.

The analysis of related work shows that no complete framework for managing ontology coherence is proposed since they do not take into account all steps of the ontology life cycle. The majority of works conducted so far in the field of ontologies has focused on ontology construction issues. These works assume that the domain knowledge encapsulated in ontology does not change over time. Indeed, in dynamic environments, the domain knowledge evolves continually due to: the evolution in the application domain, additional functionalities to add to the system, new requirements of users, needs to better organize and model the information system etc.
Most of existing systems related to the ontology evolution provide only one possibility for realizing a change, and this is usually the simplest one. For example, the deletion of a concept always causes the deletion of all its sub concepts. It means that users are not able to control the way changes are performed (supervision).
In this work, we aim to propose an evolution tool which allows taking into account all relationships and offers a great level of expression. In addition, the approaches proposed in the literature are based on the correction of inconsistencies after they occur. We propose in this paper an anticipatory approach to manage inconsistencies before they occur. We express the requirements of evolution using types of changes. For each type of change, we define corrective operations that must be applied in conjunction with this type of change in order to correct consistencies.

3. An approach based on coherent evolution kits

The identification of types of changes to apply on the ontology formally expresses the needs of evolution required by users. The types of changes allow users expressing the requirements of evolution. When they are applied, the ontology changes from a current version to another one. However, the application of a type of change can cause inconsistencies on the new ontology version. In fact, types of change ensure only the modification of ontology. They do not guarantee that the ontology remains coherent after modifications.
To ensure the consistency of an ontology after evolution, we propose to anticipate inconsistencies that can be generated by each type of change in order to propose alternatives to address these inconsistencies [Jaz09]. Thus, we defined coherent evolution kits. A coherent evolution kit is composed of a type of change and corrective operations that allow correcting the potential inconsistencies caused by the considered change. The role of corrective operations is to correct inconsistencies by proposing additional changes to be applied by the system in combination with the initial type of change required by users. If several possibilities exist, i.e., various corrective operations may be applied with different effects, the ontology engineer has to choose to implement the adequate corrective operation. Each type of change in addition to the corrective operations forms a "coherent evolution kit" that must be

applied in full. We refer to [JSG10] for more details about the evolution kits of change.

4. Consistology: a tool to ensure consistency of ontologies

In a collaborative setting, given some changes to do on the ontology, users must be able to: (1) apply changes on the ontology; (2) examine the effects of changes visually; and (3) accept or reject changes.
Due to the lack of tools providing an efficient automatic support for ontology evolution, the development of an automatic tool is very useful to maintain uniformity and consistency of ontologies. We developed the *Consistology* tool, based on Java and Eclipse, to serve as an efficient automatic support for ontology evolution. Changes on the ontology are performed using elementary and composite changes. The application of elementary and composite changes on the initial ontology allows generating a new ontology version (Figure 1).

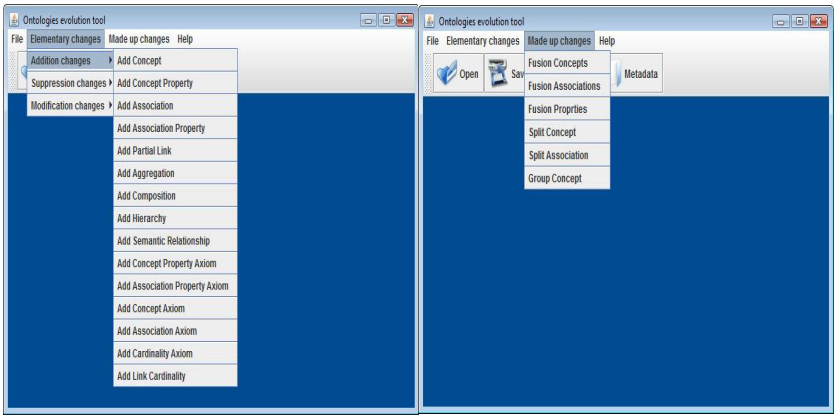


Figure 1. The Consistology tool allows applying elementary and composite changes.

The developed Consistology tool incorporates all actors (expert, ontology designer, system, user) in the evolution process. The ontology evolution process is initialized by the ontology designer and the expert, started by the user and guided by the system. The ontology designer initializes the process of evolution by introducing the ontology file and defining the metadata related to the semantic relationships. The expert defines the metadata related to the key concepts of the domain of study.

The user expresses evolution requirements using types of changes provided by the system which controls the required changes and applies the corresponding evolution kits of change in order to ensure the ontology consistency.

5. Application of Consistology to the Education domain

We present in this section an application of the developed Consistology tool to update an ontology of education related to the Tunisian higher education system.

The Tunisian higher education system is continually subject to changes to comply with social, economic and political strategies. Actually, it migrates from the old classical system toward a BMD (Bachelor's, Master's, Doctorate) system. The transition from the classical to the BMD system will certainly leave questions especially to students who followed their teachings within the old system. To provide satisfactory answers to these questions, it is necessary to understand and model the classical and the BMD systems as well as the transition between them. The modeling of this transition is also useful for the reuse of the current education system in case of future evolutions.

The modeling of the Tunisian education system requires a formal representation of knowledge. We use the ontology to explicit the semantics of the education domain and to model the classical and the BMD education systems [SJG09b]. The ontology of the BMD education system is an evolved version of the ontology related to the classical system. The evolution requires applying types of changes in order to adapt the old education ontology and to create a new ontology version adapted to the BMD system. We ensure the evolution of ontology based on primitive and complexes operators.

The acquisition of knowledge related to the education system is based on the analysis of technical documents and instruction manuals provided by the Ministry of higher education as well as interviews with experts of the domain. The ontology construction is done using *Protégé*.

We present in the following, an extract from the initial ontology of education according to the OWL syntax:

```
<owl:Ontology rdf:about=""/>
<owl:Class rdf:ID=" MEDICINE ">
  <rdfs:subClassOf>
   <owl:Class rdf:ID="SECTOR"/>
  </rdfs:subClassOf>
 </owl:Class>
<owl:Class rdf:ID="TECHNOLOGY_SCIENCES ">
  <rdfs:subClassOf>
   <owl:Class rdf:ID=" SECTOR"/>
  </rdfs:subClassOf>
 </owl:Class>
<owl:Class rdf:ID="HUMAN_SCIENCES ">
  <rdfs:subClassOf>
   <owl:Class rdf:ID=" SECTOR"/>
  </rdfs:subClassOf>
 </owl:Class>
<owl:Class rdf:ID="ECONOMICS_MANAGEMENT ">
  <rdfs:subClassOf>
   <owl:Class rdf:ID=" SECTOR"/>
  </rdfs:subClassOf>
 </owl:Class>
<owl:Class rdf:ID=" PHYSICAL_EDUCATION ">
  <rdfs:subClassOf>
   <owl:Class rdf:ID=" SECTOR"/>
  </rdfs:subClassOf>
 </owl:Class>
```

```
. .
<owl:Class rdf:ID="DOCTORATE">
  <rdfs:subClassOf>
   <owl:Class rdf ID="DIPLOMA"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="PROFESIONAL_ MASTER ">
  <rdfs:subClassOf>
   <owl:Class rdf ID="MASTER"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="MAITRISE">
  <rdfs:subClassOf rdf:resource="#DIPLOMA"/>
 </owl:Class>
 <owl:Class rdf:ID="RESEARCH_MASTER">
  <rdfs:subClassOf>
   <owl:Class rdf about="#MASTER"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="TECHNICIAN">
  <rdfs:subClassOf rdf:resource="#DIPLOMA"/>
 </owl:Class>
 <owl:Class rdf:about="#MASTER">
  <rdfs:subClassOf rdf:resource="#DIPLOMA"/>
 </owl:Class>
. .
```

To express the evolution from the classical education system toward the BMD system, we apply operators of changes such as:

1. Add new concepts which exist only in the BMD system, such as: MENTION, OPTIONAL_UNIT, COURSE, OBLIGATORY_UNIT, LICENCE, MASTER1, MASTER2, EDUCATION_UNIT etc.
2. Add new relationships between concepts such as:
   - Equivalence: for example, an equivalence relationship is added between the concepts: TECHNICIAN and LICENCE, MAITRISE and MASTER1, etc.
   - Synonymy: for example, a synonymy relationship is added between the concepts: MODULE and EDUCATION_UNIT.

6. An illustrative example

We present in this section an example of application of an evolution kit: *Add_concept*. In this example, we aim to add a new concept '*LICENCE*' to a hierarchy of concepts in the ontology of the classical education system to evolve it towards to the BMD system.

The user introduces an initial ontology to update and selects the type of change to apply on the ontology, for example *Add_Concept* (Figures 2 and 3).

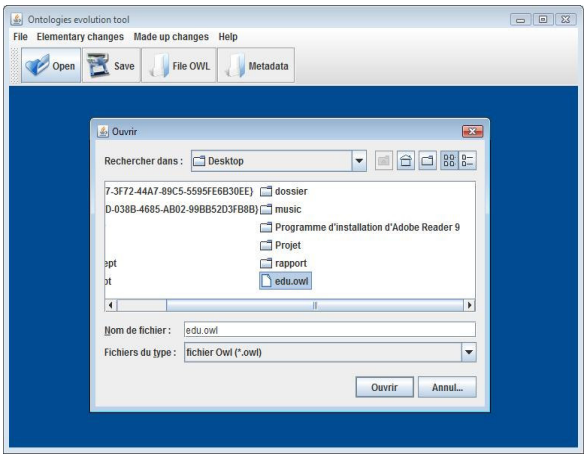


Figure 2. The input of Consistology is an ontology (e.g., owl file) to update.

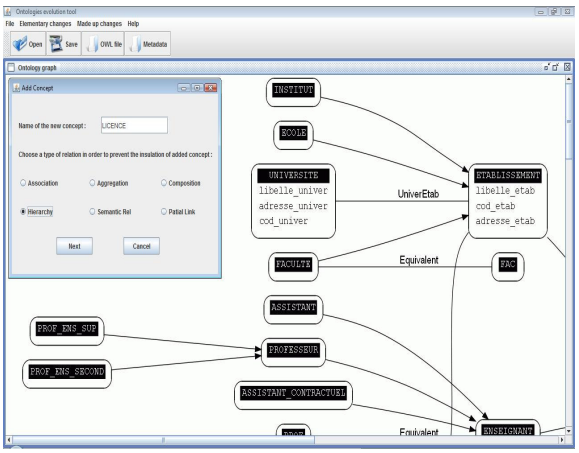


Figure 3. Add a new concept 'LICENCE' to the initial ontology.

The type of change *Add_Concept* generates inconsistencies related to an isolated and empty concept. To resolve the first inconsistency, the system automatically proposes to the user to add a new relationship between the added concept and another one in

the ontology. In this example, we chose to add a Hierarchy relationship between the concepts: *LICENCE* and *DIPLOMA* (Figure 4). Thus, since it is a hierarchy relationship, the concept *LICENCE* inherits the properties from the concept *DIPLOMA* and therefore the second inconsistency is resolved.

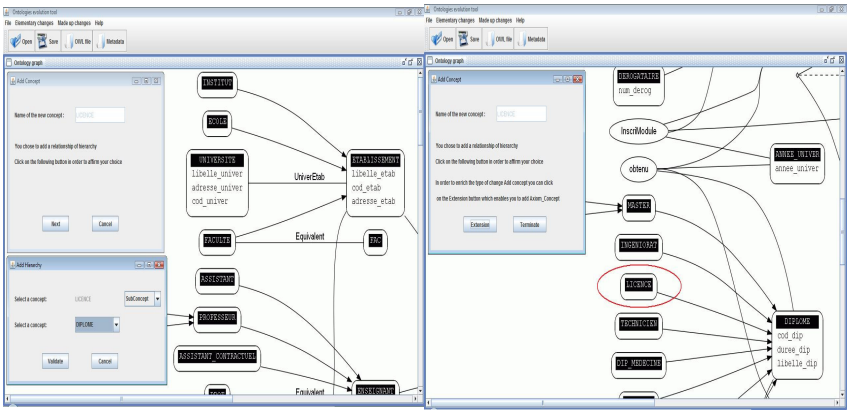


Figure 4. Add a new hierarchy relationship between the new concept LICENCE and another concept belonging to the ontology. The new concept LICENCE is added to the ontology as well as a hierarchy relationship between LICENCE and DIPLOMA.

In addition, the developed Consistology tool allows enriching the ontology by adding new axioms (Figure 5).

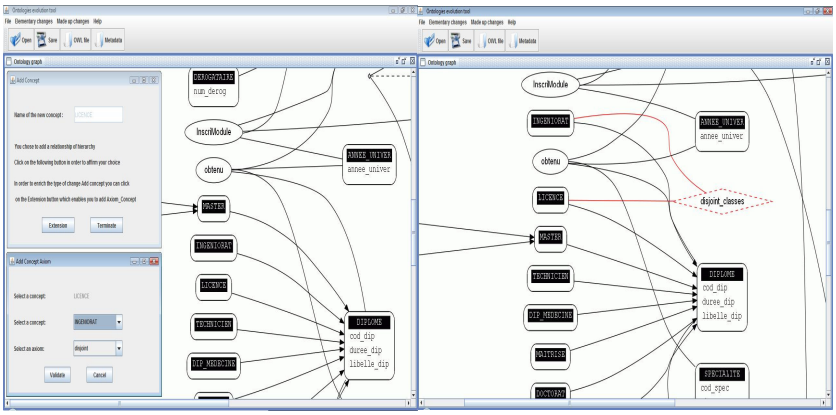


Figure 5. The tool allows enriching the ontology by adding new axioms (optional extensions). Example: A new axiom 'disjoint classes' is added between the concepts LICENCE and INGENIORAT.

The application of elementary and composite changes on the initial ontology allows generating a new ontology version (Figure 6). A historic file is created containing an ordered sequence of types of changes applied to the initial version.

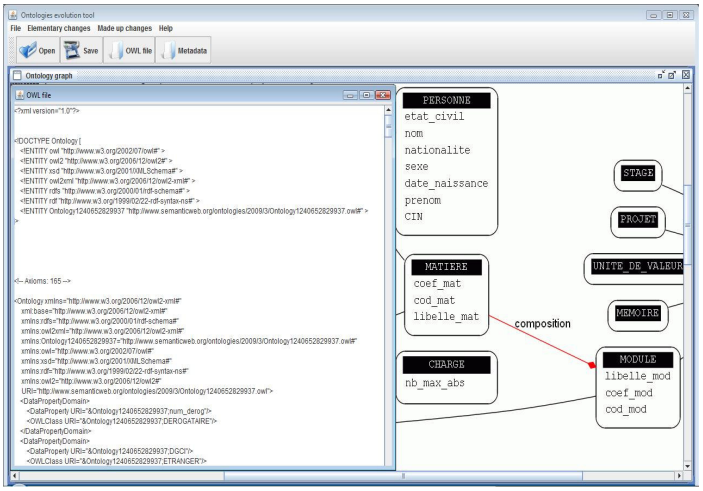


Figure 6. Consistology produces a new ontology version (OWL file) and a graph representing the ontology.

7. Conclusion and perspectives

Ontologies represent an explicit specification of a domain and serve as a support for providing and searching knowledge sources. They need to be modified to reflect new requirements and must remain coherent.

We express the requirements of evolution using types of changes. However, types of change allow updating ontology but do not ensure its consistency. The application of a type of change may produce inconsistencies on ontological entities. To correct them, corrective operations are defined and automatically done in addition to the type of changes.

An inconsistent ontology may be the consequence of a bad design or of the application of changes. We consider that the first case is rather a problem of ontology design and building. To maintain ontology consistency after applying types of changes, we developed a proactive approach to manage inconsistencies before they occur rather than managing them after evolution. This approach is based on evolution kits, defined to ensure the consistency of ontology after evolution. An evolution kit anticipates the inconsistencies that can generate each type of change in order to apply additional changes able to treat them. After the execution of a change, some corrective operations are automatically applied.

To implement types of changes, we developed the Consistology tool. Consistology is an ontology evolution support which allows users updating ontologies while preserving their consistency. It is based on elementary and composite changes that allow expressing the different possibilities of evolution requirements. Experimentation is presented, related to the evolution of the Tunisian higher education system. The *Consistology* tool is used to apply changes on the education ontology and to adapt it to new evolution requirements.
In future work, we aim to apply the developed system to other applications involving evolution changes. We will also add other functionalities to support versionning of ontology and to store and query various versions in an ontological database.
In fact, the problem of evolution and versioning is also present in other application areas, more especially in the context of databases systems. Dynamic schema evolution in databases is defined as managing schema changes in a timely manner without loss of existing data. Particular problems addressed are cascading changes (changes required to other parts of the schema as a result of a change), ensuring consistency of the schema, and propagation of the changes to the corresponding database.
Although there are significant differences between schema evolution and ontology evolution, many of the methods and technologies developed for schema evolution can be applied or adapted to ontology evolution. Our research in the ontology evolution can benefit from the many research works in database systems. Thus, we aim to exploit the techniques of databases to create versions of ontology and to incorporate additional functionalities in Consistology in order to allow representing, saving, evolving and accessing to ontology versions.

References

[BB08] Brahmia, Z., Bouaziz, R. (2008). Schema Versioning in Multi-Temporal XML Databases, Proceedings of the 7th IEE/ACIS International Conference on Computer and Information Science (IEEE/ACIS ICIS 2008), pages 158-164, Oregon, USA.

[DA10] Djedidi, R., Aufaure, M.A. (2010). ONTO-EVOAL an Ontology Evolution Approach Guided by Pattern Modeling and Quality Evaluation. FoIKS 2010: 286-305.

[Den09] Denny, M. (2009). Ontology Tools Survey. From http://www.xml.com/pub/a/2004/07/14/onto.html.

[FP05] Flouris, G., Plexousakis, D. (2005). Handling Ontology Change: Survey and Proposal for a Future Research Direction. Technical report FORTH-ICS/TR-362.

[GM03] Gómez-Pérez, A., Manzano-Macho, D. (2003). A survey of ontology learning methods and Techniques, Deliverable 1.5, Universidad Politécnica de Madrid.

[HH00] Heflin, J., Hendler, J. (2000). Dynamic Ontology on the Web, Proceedings of the Seventeenth National Conference on Artificial Intelligence AAAI/MIT, pages 443-449, CA.

[HS05] Haase, P., Stojanovic, L. (2005). Consistent Evolution of OWL Ontologies. In A. Gomez-Perez and J. Euzenat, editors, Proceedings of the 2nd European Semantic Web Conference (ESWC '05), volume 3532 of LCNS, pages 182–197. Springer.

[Jaz09] Jaziri, W. (2009). A methodology for ontology evolution and versioning, The Third International Conference on Advances in Semantic Processing (SEMAPRO'2009), pages 15-21, ISBN: 978-1-4244-5044-2, Sliema, Malta.

[JHQ+09] Ji, Q., Haase, P., Qi, G., Hitzler, P., Stadtmüller, S. (2009). RaDON: Repair and Diagnosis in Ontology Networks, Lecture Notes in Computer Science, pages 863-867, Volume 5554/2009, Springer Berlin-Heidelberg.

[JSG10] Jaziri, W., Sassi, N.,Gargouri F. (2010). Approach and tool to evolve ontology and maintain its coherence, International Journal of Metadata, Semantics and Ontologies, Inderscience Publishers, United Kingdom.

[KF01] Klein, M., Fensel, D. (2001). Ontology versioning on the Semantic Web, In Proceedings of the 1st Semantic Web Working Symposium, Stanford, CA, USA.

[KFK+02] Klein, M., Fensel, D., Kiryakov, A., Ognyanov, D. (2002). Ontology versioning and change detection on the web. Lecture Notes in Computer Science, pages 247-259, volume 2473, Springer.

[KJL09] Kroha, P., Janetzko, R., Labra, J.E. (2009). Ontologies in Checking for Inconsistency of Requirements Specification, The Third International Conference on Advances in Semantic Processing (SEMAPRO'2009), pages 32-37, Sliema, Malta.

[KPS+05] Kalyanpur, A., Parsia, B., Sirin, E., Hendler, J. (2005). Debugging unsatisfiable classes in owl ontologies, Journal of Web Semantics, volume 3(4), pages 268-293.

[LD07] Luong, P-H., Dieng-Kuntz, R. (2007). A Rule-based Approach for Semantic Annotation Evolution, The Computational Intelligence Journal, 23(3): 320-338, USA.

[MS03] Maedche A., Staab S. (2003), Ontology Learning. In S. Staab & R. Studer (eds.), Handbook on Ontologies in Information Systems, pages 173-190, Springer.

[MV01] Maedche, A., Volz, R. (2001). The Text-To-Onto Ontology Extraction and Maintenance System, Workshop on Integrating Data Mining and Knowledge Management co-located with the first International Conference on Data Mining, San Jose, California, USA.

[NM02] Noy, N., Musen M. (2002). Promptdiff: a fixed-point algorithm for comparing ontology versions. In Proceedings of the 18th National Conference on Artificial Intelligence, pages 744-750, Canada.

[NK04] Noy, N., Klein, M. (2004). Ontology evolution: Not the same as schema evolution, Knowledge and Information Systems, 6(4):428-440.

[OVM+04] Oberle, D., Volz, R., Motik, B., Staab, S. (2004). An extensible ontology software environment, In: Steffen Staab, Rudi Studer (Eds.), Handbook on Ontologies, pages 311–333, Springer, Berlin.

[RL05] Robbes, R., Lanza, M. (2005). Versioning systems for evolution research, In Proceedings of the 8th International Workshop on Principles of Software Evolution (IWPSE 2005), pages 155-164, Lisbon, Portugal.

[SBJ+10] Sassi, N., Brahmia, Z., Jaziri, W., Bouaziz, R. (2010). From Temporal Databases to Ontology Versioning: An Approach for Ontology Evolution, Ontology Theory, Management and Design: Advanced Tools and Models, Ed. Faiez Gargouri and Wassim Jaziri, IGI-Global, USA (to appear, March 2010).

[SC03] Schlobach, S., Cornet, R. (2003). Non-standard reasoning services for the debugging of description logic terminologies. In Proceedings of the 18th International Joint Conference on Artificial Intelligence, pages 355-362, Acapulco, Mexico.

[SSG+03] Stojanovic, L., Stojanovic, N., Gonzalez, J., Studer, R. (2006). Ontomanager - a system for the usage-based ontology management", In Proceedings of ODBASE'2003, pages 858-875, Springer.

[Sto04] Stojanovic, L. (2004). Methods and Tools for Ontology Evolution. PhD Thesis, University of Karlsruhe.

Previewing OWL Changes and Refactorings Using a Flexible XML Database

Christoph Lange and Vyacheslav Zholudev

Computer Science, Jacobs University Bremen,
{ch.lange,v.zholudev}@jacobs-university.de

Abstract. During their lifecycle, ontologies are changed for diverse reasons: their vocabulary is enhanced to enable additional application or annotation possibilities, their expressivity is restricted to speed up reasoning, their internal structure is refactored for alignment with other ontologies or to facilitate maintenance, and many more. Any such change can have serious consequences on applications using an ontology; therefore it has to be done with care. TNTBase is a versioned XML database supporting *virtual documents*: XQuery-based views on XML documents that appear to the user as files. We use this feature in order to preview changes to OWL 2 XML ontologies: any proposed change is first tested in a virtual document, before it is applied to the actual ontology. We demonstrate the flexibility of this approach in several cases of changes, discuss the limitations of working with ontologies on XML level, and propose an integration of TNTBase as a backend with ontology editors.

1 Introduction

Change management and refactoring are important parts of the ontology lifecycle. Originally investigated in software engineering, they are now also gaining more and more attention and software support in ontology engineering (see, e. g., [19, 6]), often within the larger context of ontology evolution [25]. Change management for ontologies has been defined as "the process of performing the changes as well as [...] the process of coping with the consequences of changes" [12]. Refactoring in software engineering is commonly defined as "a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior" [7]; this definition is also valid for ontologies. Typical ontology refactorings include splitting an ontology into several modules, or, conversely, merging multiple ontologies, moving axioms to another module of the same ontology, or rewriting axioms to semantically equivalent but shorter or longer forms (e. g., in description logics, rewriting the two axioms $A \sqsubseteq B, A \sqsubseteq C$ into $A \sqsubseteq B \sqcap C$). Despite the strict definition of refactoring, practical development environments, including Protégé, also subsume restructurings that do change the external behavior of code under "refactoring", e. g. changing the URI of an ontology or entity [6]. However, well-behaved tools usually try to keep the potential damage done by such a change as low as possible by adapting all ontologies in the current project to it. On the other hand, it is not trivial for a

development environment to estimate the full impact of a change, which means that in practice changes often *do* break other things, such as other ontologies based on the current one, ontology-based software implementations, documents annotated with terms from ontologies, etc. Consider the axiom refactoring example mentioned before, and suppose that A, B, and C are not concepts but roles. Not all description logics support role intersection, so certain reasoners might not support the supposedly equivalent rewriting of two role inclusions into one.

Therefore, it is safe to assume that *any* change may break things. When multiple developers collaborate in a shared repository, this means changes should be made with care, and that they should be tested. We present a repository system that supports testing changes by not making them to the physical ontology files in the first place, but by creating them as *views* on the unchanged physical files. We have implemented this on top of our versioned XML database TNTBase for ontologies in the XML encoding of OWL 2. We discuss common change and refactoring cases and show their realization in TNTBase, also taking larger engineering workflows as a part of the ontology lifecycle into consideration. At the same time, this is an experiment in exploring the advantages and limits of managing OWL ontologies on the level of XML document collections.

2 TNTBase, a Versioned Database for XML

TNTBase is a versioned client-server XML database [27]. Essentially, it consists of two parts: the core and the application-specific layer. Let us briefly discuss the technical foundations we need for refactoring OWL 2 XML ontologies.

2.1 The Core

The xSVN module, which integrates Berkeley DB XML [1] into a Subversion server [23], is the core of TNTBase. DB XML stores HEAD revisions of XML files; non-XML content like PDF files or images, differences between revisions, directory entry lists and other repository information are retained in a usual SVN back-end storage. Keeping XML documents in DB XML allows us to access those files not only via any SVN client, but also through the DB XML API that supports efficient querying of XML content via XQuery [2] and modifying it via XQuery Update [3]. As many XML-native databases, DB XML also supports indexing, which improves performance of certain queries.

TNTBase is realized as a web-application with two different communication interfaces: an xSVN interface and a RESTful interface for XML-related tasks. The xSVN interface behaves like the normal SVN interface – Apache's *mod_dav_svn* module serves requests from SVN clients – with one exception: When an ill-formed XML file is committed, xSVN aborts the whole transaction. The RESTful [11] interface provides XML fragment access to the versioned collection of documents:

Querying: As every XML-native database, TNTBase supports XQuery, but extends the DB XML syntax by a notion of file system paths to address path-based collections of documents.

Modifying: Apart from modifying documents via an SVN client, TNTBase takes advantage of XQuery Update facilities, and, in contrast to pure DB XML, versions modified documents: A new revision is committed to xSVN whenever a documents in a TNTBase repository has been changed.

Querying previous revisions: Although xSVN's DB XML back-end by default holds only the latest revisions of XML documents, and others are stored as reverse deltas against them, it is possible to access and query previous versions by additionally providing a revision number to the TNTBase XQuery extension functions. However, previous versions cannot be modified since once a revision is committed to an xSVN repository, it becomes persistent.

2.2 Application Layer of TNTBase

It turned out that many tasks specific to particular XML formats can be done by TNTBase, and that was a reason to derive a separate layer on top of the TNTBase core and augment this layer with format-specific functionalities. Detailed information can be found in [28], but let us briefly describe the major features:

Virtual Documents are essentially "XML database views" analogous to views in relational databases; these are tables that are virtual in the sense that they are the results of SQL queries computed on demand from the explicitly represented database tables. Similarly, TNTBase virtual documents (VD) are the results of XQueries computed on demand from the XML files explicitly represented in TNTBase, presented to the user as entities (files) in the TNTBase file system. Like views in relational databases, TNTBase VDs are editable, and the TNTBase system transparently patches the differences into the original files in its underlying versioning system. Thus, a user does not have to know about the original source of document parts and it allows him to focus only on relevant pieces of information. Again, like relational database views, VD become very useful abstractions in the interaction with versioned XML storage. Technically, VDs are document templates with embedded references to any number of XQueries and instructions how to fill those templates with XQuery results. For example, a VD for a table of contents may consist of a caption, some author information and XQuery that selects all chapter titles from a collection of documents. For more examples and understanding, check out [29]. Furthermore, VDs can be addressed directly in XQuery by calling a dedicated XQuery TNTBase function. Hence they may seemlessly be used in users' XQueries as well as comprise the content of other VDs.

Validation and Presentation: TNTBase provides facilities to integrate format-specific validation and presentation. Simple examples are XSL transformations and schema validation. But sometimes a user needs even more, e.g. extract RDF information upon commit and cache it or retrieve a human-readably rendered representation of a document in semantics-oriented markup. Utilizing the TNTBase plugin API one can write additional modules and inject them into the application layer. Configuration files are also stored in a TNTBase repository, so a user do not require access to the server. Commit-time behavior is defined by the SVN *tntbase:validate* property that can be

assigned to files as well as to whole directories. Pre-commit or post-commit hooks that are automatically generated take care of processing committed information based on the configuration files. In case of pre-commit processing a corresponding plugin has access to the documents that are about to be committed, and may reject a transaction if the collection of committed documents is format-inconsistent, or clashes with existing documents in the repository. Last but not least, URLs that are used to perform validation or presentation are dynamically changed once configuration files are modified.

Custom XQuery modules: A user can write his own XQuery extensions and store them in the repository. Thus it is not necessary to have modules located in the server's file system or remotely. XQuery modules can be referenced inside repository itself, which is particularly useful if the development of XQuery modules is still in progress.

3 Previewing Changes and Refactorings in TNTBase

As the main contribution made by this paper, we will analyze how a number of common OWL ontology change and refactoring tasks can be realized in TNTBase by means described in section 2. As TNTBase is an XML database, we have to agree on a fixed XML schema for our implementation. We chose OWL 2 XML [15], the straightforward XML encoding of the functional-style syntax [17] of OWL 2, in terms of which the direct semantics is specified [16]. The RDF/XML serialization of OWL is still more widely used but has to sacrifice the elegance of the functional-style syntax to accommodate for restrictions imposed by RDF's graph-based data model. Moreover, RDF/XML is a particularly awkward RDF serialization, as it cannot be validated against an XML schema, and as there are too many alternatives for expressing the same facts (cf. [5]). Committing to the OWL 2 XML serialization is not a conceptual restriction, though, as translations from and to other serializations can be provided independently, e. g. by the OWL API [24]. Thanks to the Subversion-based infrastructure, such a translation can even be integrated almost transparently into TNTBase: A post-commit hook could translate any committed RDF/XML file to OWL XML.

We assume that the ontologies one wants to work with are stored as OWL 2 XML files in a TNTBase repository.[1] The file and directory structure is completely up to the users. We will not *perform* changes in the first place, but *preview* them. Change and refactoring patterns will be implemented as VDs, which can be applied to any existing ontology – an actual physical OWL XML document, or another VD. The latter permits chaining multiple change steps. Our VDs will usually affect relatively small pieces of an OWL XML document, while leaving the rest unchanged. Therefore, their implementation will heavily

[1] We provide an entry point for a sandbox repository at `https://tntbase.org/wiki/usecase_ontologies`, where you can apply the refactorings presented below, or your own ones, to your ontologies.

rely on the XQuery update facility [3], which allows for concisely expressing changes to XML documents.[2]

We discuss some typical change and refactoring patterns below. In each case, we give a general description, then describe its implementation for OWL XML, and discuss its impact the current ontology or other ontologies depending on the current one. Changes with a local impact do not affect dependent ontologies, but they may affect the *usage* of the current ontology – e. g. if a syntactic construct is introduced that the reasoner or editor in use does not support. Refactorings in the strict sense of the definition are local – recall the definition in section 1 –, but many changes commonly considered "refactorings" – such as renamings – have non-local effects on dependent ontologies or annotated resources. The larger the impact of a change becomes, the more desirable do we consider evaluating the impact by previewing the change using a VD in TNTBase.

3.1 Renaming Entities

All entities of an OWL ontology – classes, properties, and individuals – are identified by IRIs. Renaming entities is a frequent refactoring task. Such a change affects the ontology O in which an entity is declared, all ontologies importing O, and ultimately all external documents or software using O. The refactoring VD not only has to be applied to O, but to all dependent ontologies. We restrict our investigations to non-distributed repositories and thus to dependencies within the same repository. We have not yet automated the cross-document part of this refactoring; one would have to manually apply the VD to all ontologies in the repository.[3]

IRIs can be absolute or relative. Relative IRIs are resolved against the base IRI, which defaults to the URI of the ontology document but can be changed using the *xml:base* mechanism. Absolute IRIs can be abbreviated using a *prefix:localname* syntax, where prefixes are defined on the top level of the ontology, e. g. `<Prefix name="foaf" IRI="http://xmlns.com/foaf/0.1/"/>`. Renaming such abbreviated IRIs is still quite straightforward, compared to the RDF/XML serializations of OWL, which delegates prefix↦IRI mapping to the more involved namespacing mechanism of XML.

3.2 Factoring out Modules

When an ontology grows large and hard to manage, developers often identify modules and factor them out into subontologies. This is, for example, supported by the Module Extraction plugin of the NeOn Toolkit [18]. Common candidates for such subontologies are all sub- or superclasses of a given class, possibly with their instances, with related properties, and other dependent entities. The subontology S factored out should be connected to the original ontology O via an

[2] The transform functions of XQuery Update, which we use here for clarity, are not yet completely reliable in DB XML. Therefore, our actual implementation explicitly recurses over XML trees, which is less elegant.

[3] In a well-formed collection, applying a VD to *all* ontologies will not do any harm.

import. Either direction of the import could make sense: (i) If O is intended to be the main ontology reused by other ontologies and applications, O should import S. Here, O just happens to have a modular structure *internally*, but applications need not know that. (ii) Conversely, O can be designed as a *core ontology*, of which S is a domain-specific refinement. Applications in the respective domain would rather use S, which imports O. We have implemented XQuery functions for selecting certain subontologies, such as the subclasses of a given class, and generic functions that factor out a given subontology. Previewing such a change requires two VDs: one for O – removing S from it and possibly replacing it by an import link –, and one for the new ontology document containing S.

3.3 Merging Modules

The inverse operation to modularization is merging several modules back into one, which can be desirable for deploying an ontology, or for compatibility with tools that do not support modular ontologies. Here, we will only consider the easy case that all modules to be merged are disjoint, i. e. that no two modules declare two different entities with the same IRI. Then, merging reduces to merging axioms and removing import links. Multiple ontology modules spread over different files and directories can be addressed by the *tnt:collection* XQuery function. For instance, a part of a query `collection('/onto*//*.owx')` will address all OWL XML ontologies in the child folders of directories having the `onto` prefix.

3.4 Rewriting Axioms

In section 1, we have seen a case of rewriting axioms in a semantics-preserving way. In the migration to OWL 2, there are more such cases. OWL 2 not only introduces new axiom or construct types that require additional expressivity, but also mere syntactic sugar [9]. A prominent example for that is the disjoint union of classes. In OWL 1 one had to state separately the pairwise disjointness of a group of classes $D_1, \ldots, D_n$, and the equivalence $D = D_1 \sqcup \cdots \sqcup D_n$.[4] The *DisjointUnion* axiom of OWL 2 allows for directly stating that D is the disjoint union of $D_1, \ldots, D_n$. In order to make legacy ontologies from the OWL 1 age more readable, or in order to benefit from performance improvements offered by OWL 2 reasoners, it is thus desirable to rewrite disjoint unions. This is a change with a local impact. The XQuery in listing 1.1 rewrites separate declarations of pairwise disjointness and union equivalence into single disjoint union axioms. Listing 1.2 shows how that XQuery can be used for creating a VD specification. Note that we factored out the query itself to a separate module and only reference it from a specification by providing a repository-scoped URI. In section 4, we will

[4] In the RDF/XML syntax, which was the most common one for OWL 1, there was not even a shorthand notation for expressing pairwise disjointness of more than two classes. In the following, we will, however, not deal with translations between RDF/XML and OWL XML, but assume that that has been done before by a lower-level tool.

see how to apply this VD specification to a concrete input document (for more information on what VD specifications are and how to use them refer to [26, 29])

Listing 1.1. Creating disjoint union axioms

```
module namespace tntx = "http://tntbase.mathweb.org/ns/ores";
declare function tntx:refactor-disjoint-union(
 $doc as document-node()) as document-node() {
  copy $tmp := $doc modify (
    (: find equivalent class declarations with two children :)
    for $equiv in $tmp/owl:Ontology/owl:EquivalentClasses[count(*) eq 2
     and owl:ObjectUnionOf                          (: ... one of which is a union :)
     and *[not(self::owl:ObjectUnionOf)]] (: ... and the other one is something else :)
    let $whole := $equiv/*[not(self::owl:ObjectUnionOf)]                      (: D :)
    let $parts := $equiv/owl:ObjectUnionOf/*                        (: D1, ..., Dn :)
    (: look for declarations of pairwise disjointness of D1, ..., Dn :)
    for $disjoint in $equiv/../owl:DisjointClasses[fn:deep-equal(*, $parts)]
    return (
      delete node $disjoint,    (: delete the disjointness axiom and replace the ... :)
      replace node $equiv with    (: ... equivalence axiom by a disjoint union axiom :)
        <owl:DisjointUnion> { $whole, $parts } </owl:DisjointUnion> )
  ) return $tmp };
```

Listing 1.2. VD specification for disjoint union axioms

```
<tnt:virtualdocument xmlns:tnt="http://tntbase.mathweb.org/ns">
  <tnt:skeleton>
    <Ontology xmlns="http://www.w3.org/2002/07/owl#">
      <tnt:xqinclude>
        <tnt:query name="disj.xq"/>
        <tnt:return><tnt:result/></tnt:return>
      </tnt:xqinclude>
    </Ontology>
  </tnt:skeleton>
  <tnt:query name="disj.xq">
    import module namespace tntx = 'http://tntbase.mathweb.org/ns/ores'
       at 'tntbase:/modules/refactor/disjoint-union.xq';
    tntx:refactor-disjoint-union(tnt:doc($ontology-path))
  </tnt:query>
  <tnt:params><!-- parameter declarations, with default values that can be overridden -->
    <tnt:param name="ontology-path">    <!-- on creating a VD from this specification -->
      <tnt:value>/ontologies/test-ontology.owx</tnt:value>
    </tnt:param>
  </tnt:params>
</tnt:virtualdocument>
```

3.5 Lowering Expressivity

OWL 2 has several sub-profiles [14]. They allow for efficient reasoning, as they correspond to less expressive logics than $\mathcal{SROIQ}$, which covers the full expressivity of OWL 2. There are $\mathcal{SROIQ}$ reasoners, but reducing the expressivity of

an ontology may be desirable in order to benefit from a more efficient reasoner. For example, the "QL" profile can be implemented on top of an SQL database. If an existing ontology is more expressive than the desired profile, certain complex axioms and constructs will have to be removed. Among the axiom types that OWL 2 QL does not support, there are declarations of functional, inverse functional, and transitive properties. These are easy to identify, as they are represented by XML elements on the top level of the ontology. Some constructs are not allowed in certain places, such as existential quantification in the subclass position (e. g. $\exists R.C \sqsubseteq D$). Other constructs, such as unions of classes, are completely forbidden. When lowering the expressivity of an ontology, some forbidden axiom types, or axioms containing forbidden constructs have to be stripped entirely, whereas others can be weakened. Stripping is easy to implement using XQuery Update; one would simple delete nodes that satisfy a certain node test. Weakening usually requires adding a number of new axioms to the ontology. For example, the union $\bigsqcup_{i=1}^{n} C_i$ is the smallest common superclass of $C_1, \ldots, C_n$ and can therefore be weakened to a class C with $C_i \sqsubseteq C$ for each $i = 1, \ldots, n$. Another forbidden construct is the complement of a class in subclass expressions, e. g. $\neg A \sqsubseteq B$. This can be weakened by introducing a $C \sqsubseteq B$, where $A \sqcup C \sqsubseteq \bot$, i. e. where A and C are disjoint, which is allowed in OWL 2 QL.

3.6 Stripping Axiom Annotations

OWL 2 introduced annotation of axioms, not just of entities of an ontology. Axiom annotations do not yet enjoy wide tool support. For example, Protégé supports them, whereas the NeOn Toolkit doesn't. They are particularly cumbersome to handle when an OWL ontology is represented in RDF (cf. [21]). Thus, we have implemented a change pattern that strips annotations from axioms. Fortunately, all kinds of annotations are easy to handle in the XML syntax. They are simply given as an optional sequence of initial child elements of an axiom:

```
<ClassAssertion>
  <Annotation>
    <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
    <Literal>TNTBase is a database</Literal>
  </Annotation>
  <Class abbreviatedIRI="dbpedia:Database"/>
  <NamedIndividual IRI="http://tntbase.org/tntbase"/>
</ClassAssertion>
```

3.7 Common Query Patterns

We realized that some functions are reused in many ontology refactoring XQueries. Thus, we distilled them into a shared XQuery module that includes common auxiliary functions. The most common ones are: getting the base IRI of a node, or expanding an IRI using the prefix information and the base IRI in the scope (cf. section 3.1). With standard XQuery – without the update facility –, recursing

over an XML tree and returning subtrees with some changes applied would be another recurring pattern, which can be implemented by a recursive function. The most involved function we have implemented so far is equality: Reducing equality of a set of OWL expressions to equality of XML trees (*fn:deep-equal*), as done in listing 1.1, is a bit naïve, as it hardly takes into account the semantics of OWL. For example, in many constructs, such as *DisjointClasses* or *ObjectUnionOf*, the order of class expressions does not matter. Other cases of semantic equivalence of syntactically different expressions are related to IRIs (cf. section 3.1), which can be written absolutely, relatively, or using prefixes. Therefore, we have implemented OWL expression equality as a recursive XQuery function, which defaults to XML tree equality but handles other cases specially.

4 Integration into the Ontology Lifecycle

Over its whole lifecycle, an ontology will change many times, requiring extensive work with VDs. Applying a single change is straightforward. First, to create a VD one just has to give it a name and associate it with a VD specification path. In this step, query parameters can be associated with a VD; for example, the ontology path parameter in listing 1.2 can be overridden to point to the actual ontology the user wants to refactor. Then content retrieval is done by providing a path of a created VD. In XQueries, e. g. in other VDs, the content of a VD can be addressed by the *tnt:vd* function. From a user perspective, getting a VD is the same as getting a usual file. There is one exception, though: VDs can be obtained only via the RESTful interface of TNTBase. In addition, dynamic parts of a VD (i. e. parts that are results of VD XQueries) are editable. When a user modified a VD, he can commit it back using a RESTful method or a special XQuery function *tnt:submit-vd*. All changes will be propagated to the original documents [28]. Another feature of the RESTful interface is *materializing* of VDs – when refactoring is finished and the content of a VD looks right[5], a user is able to create a refactored ontology as a physical document in the TNTBase repository based on VD content. Alternatively, the possibility of having editable VDs does not force ontology engineers to materialize all changes: Different developers can keep different views on (sub)ontologies and work with them, without having to adapt to a particular structure of the actual physical ontology.

Our approach should scale well to large ontologies. DB XML indexing facilities might tremendously reduce query time. For instance, we experimented with a collection of 2000 documents and ran filtering queries based on attribute and element names. Adding indexes reduced the timing from 30 seconds to 0.5 seconds. Whereas we do not claim that such speed improvement will be achieved for every query, it gives us a better impression how things may scale in TNTBase. Multiple users can read/write to TNTBase simultaneously – TNTBase secures every user action with a transaction and takes care about deadlocks resolution.

[5] TNTBase itself does not support the collaborative decision making that is required here. We leave that feature to a future integration of TNTBase into a development environment that, e. g., supports argumentation.

5 Related Work

Several ontology editors offer client-side refactoring. Protégé natively supports a few basic refactorings [6], whereas for the NeOn toolkit several more sophisticated refactoring plugins have been developed [18]. In contrast, our solution is, to the best of our knowledge, the only one that supports ontology refactoring in a server-side repository. Once a VD specification has been provided for change pattern, any other user can apply it to any ontology and load the result into his favorite development environment for evaluation and testing. We believe that the overall workflow would benefit from a closer client-server integration. The NeOn toolkit seems the most suitable candidate for an integration of our work, as its underlying Eclipse IDE has a Subversion client built in. We would only have to add support for those features that require interaction with TNTBase's RESTful interface, such as the creation or materialization of VDs. While we have focused on changes and refactorings, the Evolva framework – implemented as another NeOn plugin – addresses the general challenge of ontology evolution [25]. It validates an ontology after each change, checking for consistency, duplication, and time-related inconsistencies. TNTBase can perform schema validation on materialized VDs; more advanced validation can be done by integrating 3$^{\text{rd}}$-party validation plugins. All in all, Evolva dynamically adapts an ontology to a changed environment. We believe that such an evolution framework could be nicely complemented with our database backend. Finally, there are alternatives to querying OWL 2 XML with XQuery (Update): for example, representing an OWL ontology as RDF, querying it with SPARQL, and changing it using the proposed SPARQL-Update [22]. A clear specification of SPARQL under different entailment regimes, such as OWL, is under way [8], but the implications on SPARQL-Update have not yet been investigated. RDF is on a higher abstraction level than XML, which practically means, for example, that different syntaxes of encoding IRIs (cf. section 3.1), which make a difference in XQuery, do not affect a SPARQL query. But the RDF encoding of OWL is contrived in other aspects: Everything has to be broken down to RDF triples, which introduces artificial complexity for n-ary structures that can be represented in a straightforward way in XML. The new SPARQLAS language, however, supports intuitive query formulation in the OWL functional-style syntax [20], which is internally translated to standard SPARQL. So far, it only covers querying ontologies, though, not updating them. The OWL API [24], the technical basis of Protégé and the NeOn toolkit, does not change ontologies by queries but programmatically by Java methods, but makes it convenient to implement refactoring algorithms, as all OWL constructs are represented as Java objects on a semantic level, abstracting from a concrete serialization. On the other hand, the OWL API has to parse a complete ontology into the memory, whereas Berkeley DB XML, the database underlying TNTBase, does not have to do that, and therefore is more scalable.[6]

[6] Actually, the OWL API is prepared for ontologies stored in databases (cf. [10]). Out of the box it only provides an in-memory representation, but integrating it with a TNTBase backend would be feasible.

6 Conclusion and Future Work

We have showed how TNTBase, a versioned XML database, supports ontology changes and refactorings. As changes and refactorings can break modules or resources that depend on an ontology, we do not immediately apply them, but create them as views on the original ontology, using TNTBase's virtual documents. We have showed that several common change patterns can be implemented for OWL XML using XQuery at a reasonable cost – even more so now that we provide a module of commonly needed functions. Thanks to XQuery Update, changes can be written down concisely. Applying them to given ontologies is straightforward as long as the ontologies are implemented in single files; otherwise more work is required, which could be automated in future, though. As OWL XML is a direct XML encoding of the OWL functional-style syntax, in terms of which the direct semantics of OWL has been specified, processing on XML level is surprisingly close to processing it on a higher "semantic" level. However, we initially had to implement some of the OWL semantics in XQuery, such IRI expansion and the somewhat more involved equality of expressions. Now, these are part of a reusable XQuery module, to which we will add further functionality, possibly including functions that compute ontology metrics.

Here, we have focused on the repository management and XML querying features of TNTBase, but TNTBase can actually do more. For different document formats, we have shown how to extend TNTBase by XML→RDF translations that are automatically run when committing an XML file, how to integrate an RDF triple store, and how to serve Linked Data – raw RDF, or RDFa embedded into human-readable XHTML documents (think of ontology documentation) [4, 13]. We consider this a beneficial complement to the OWL change and refactoring functionality here. If the OWL ontologies are in a triple store with a reasoner attached, more sophisticated SPARQL(AS) queries will become possible.

With change patterns and ontologies stored in the same repository, TNTBase enables an agile way of collaborative refactoring, where a development team can not only discuss the outcome of a refactoring step, but also easily improve the change patterns. We will explore the potential of this methodology in further case studies with real-world ontologies. So far, implementing a change pattern involves manual XQuery editing, and instantiating requires manual VD administration in the repository. These tasks could be facilitated for ontology engineers by a closer integration of TNTBase with a client-side development environment. The NeOn toolkit can already access Subversion-compatible repositories, and Protégé has been successfully connected to other kinds of repositories. Therefore, a closer TNTBase integration seems feasible, and would take us closer to the goal of editable ontology repositories.

References

[1] *Berkeley DB XML*. URL: `http://oracle.com/database/berkeley-db/xml/`.

[2] *XQuery 1.0: An XML Query Language*. Recommendation. W3C, 2007.

[3] *XQuery Update Facility 1.0*. Candidate Recommendation. W3C, 2009.
[4] C. David, M. Kohlhase, C. Lange, F. Rabe, N. Zhiltsov, and V. Zholudev. "Publishing Math Lecture Notes as Linked Data". In: *ESWC*. 2010. arXiv: 1004.3390.
[5] I. Davis. *The Sixteen Faces of Eve*. 2005. URL: http://iandavis.com/blog/2005/09/the-sixteen-faces-of-eve.
[6] N. Drummond. *Protege 4.x Menu Actions and Keyboard Shortcuts*. 2009. URL: http://protegewiki.stanford.edu/index.php?title=Protege4Shortcuts&oldid=6142#Refactor.
[7] M. Fowler. *Refactoring Home Page*. URL: http://refactoring.com.
[8] *SPARQL 1.1 entailment regimes*. Working Draft. W3C, 2009.
[9] *OWL 2: New Features and Rationale*. Recommendation. W3C, 2009.
[10] M. Horridge and S. Bechhofer. "The OWL API: A Java API for Working with OWL 2 Ontologies". In: *OWLED*. 2009.
[11] *JSR 311: JAX-RS: The Java API for RESTful Web Services*. URL: https://jsr311.dev.java.net/nonav/releases/1.0/index.html.
[12] M. Klein. "Change Management for Distributed Ontologies". PhD thesis. Vrije Universiteit Amsterdam, 2004.
[13] C. Lange and M. Kohlhase. "A Mathematical Approach to Ontology Authoring and Documentation". In: *MKM*. Springer, 2009.
[14] *OWL 2: Profiles*. Recommendation. W3C, 2009.
[15] *OWL 2: XML Serialization*. Recommendation. W3C, 2009.
[16] *OWL 2: Direct Semantics*. Recommendation. W3C, 2009.
[17] *OWL 2: Structural Specification and Functional-Style Syntax*. Recommendation. W3C, 2009.
[18] *NeOn Toolkit*. URL: http://neon-toolkit.org.
[19] D. A. Ostrowski. "Ontology Refactoring". In: *IEEE International Conference on Semantic Computing*. IEEE, 2008.
[20] F. S. Parreiras et al. *SPARQLAS*. URL: http://code.google.com/p/twouse/wiki/SPARQLAS.
[21] *OWL 2: Mapping to RDF Graphs*. Recommendation. W3C, 2009.
[22] *SPARQL 1.1 Update*. Working Draft. W3C, 2009.
[23] *Subversion*. URL: http://subversion.tigris.org/.
[24] *The OWL API*. URL: http://owlapi.sourceforge.net.
[25] F. Zablith, M. Sabou, M. d'Aquin, and E. Motta. "Ontology Evolution with Evolva". In: *ESWC*. 2009.
[26] V. Zholudev and M. Kohlhase. "Scripting Documents with XQuery: Virtual Documents in TNTBase". In: *submitted to Balisage: The Markup Conference*. 2010. URL: http://kwarc.info/kohlhase/papers/balisage10.pdf.
[27] V. Zholudev and M. Kohlhase. "TNTBase: a Versioned Storage for XML". In: *Balisage: The Markup Conference*. Vol. 3. Mulberry, 2009.
[28] V. Zholudev, M. Kohlhase, and F. Rabe. "A [insert XML Format] Database for [insert cool application]". In: *XML Prague*. 2010.
[29] V. Zholudev et al. *TNTBase – Virtual Documents*. 2010. URL: http://trac.mathweb.org/tntbase/wiki/VD.

On the Use of Transformation and Linked Data Principles in a Generic Repository for Semantic Web Services

Barry Norton[1], Mick Kerrigan[2], and Adrian Marte[2]

[1] AIFB, Karlsruhe Institute of Technology, Germany
barry.norton@kit.edu
[2] Semantic Technology Institute, University of Innsbruck, Austria
first.last@sti-innsbruck.at

Abstract. As yet, despite many years of research into Semantic Web Services, there is no standard Semantics-based service model. SA-WSDL, a W3C recommendation, provides links only to an unspecified semantic model from WSDL, itself merely an XML syntax. Ontology-based approaches such as OWL-S and WSMO have never advanced beyond the stage of submission to standards bodies. In this environment one approach to moving forward is exemplified by WSMO-Lite, which can be viewed as an 'intersection' of the features of OWL-S and WSMO as applied a semantic representation of the relevant structure of WSDL, the so-called 'minimal service model'. A relatively newer standards submission, named the Semantic SOA Reference Ontology and the subject of this paper, takes instead a 'union' approach to the features considered so far. We shall show that among the advantages of such an approach is its ability to mediate between the existing models. The major contribution of this paper, alongside detailing the Reference Ontology, is to show how this mediation can be effected simply using SPARQL and exemplify the practicality of this solution as the basis for a RESTful service repository based on Linked Data principles.

1 Introduction

Semantic Web Services (SWS) provide a means for creating richer descriptions for Web Services, where explicit ontology-based semantics increase automation in service creation and consumption tasks via reasoning. Semantic Web Services form a layer on top of existing Web Service technologies and not a replacement for them.

In order to use the semantic descriptions present in a so-called Semantic SOA (SSOA), to automate the tasks associated with Service-Oriented Architectures (SOA), a set of reasoning-based platform services are required within the SSOA. These services are collectively termed a Semantic Execution Environment (SEE) and form the core of a SSOA-based implementation. There are a number of different implementations of SEEs currently under development in the research community, which have some common features. Examples of such Semantic Execution Environments are WSMX [1], IRS-III [2], and METEOR-S [3].

The OASIS Semantic Execution Environment Technical Committee (SEE-TC)[1], which is co-chaired by the authors, was established in 2005, with the aim to:

> "provide guidelines, justifications and implementation directions for an execution environment for Semantic Web services. The resulting infrastructure will incorporate the application of semantics to service-oriented systems and will provide intelligent mechanisms for consuming Semantic Web services."

The SEE-TC is in the process of standardizing the types of platform services that exist within a Semantic Execution Environment for Semantic Web Services, their black box behaviour, and their interfaces. In order to consistently define these platform service the SEE-TC have first defined a Semantic SOA Reference Ontology (SSOA-RO), which provides a description of the elements that need to be modeled in order to effectively provide semantic description for services.

Currently under public review, on the route towards OASIS standardisation, one of the main requests was that the Reference Ontology be made available in the form of RDFS. Work towards this has opened up the possibility, motivated particularly the SEALS project[2], the ability to transform descriptions in existing service models into the SSOA-RO, and to use this as the basis to produce definitions in these different models. It has been found that this is largely feasible using the 'CONSTRUCT' syntax for the semantic query language SPARQL.

The authors specified, for the SEALS project, which is concerned with the benchmarking and evaluation of semantic technologies, RESTful APIs for repositories containing tools, test data (including SWS descriptions), and evaluation results[3]. The transformation of service descriptions into different service models was included as an instance of 'synthetic test data generation', where descriptions stored in the Semantic SOA Reference Ontology are transformed via SPARQL.

In the meantime, the SOA4All project [4] has pointed out that such a RESTful API for the management of such descriptions, where the descriptions are exposed in RDF as they are in the SEALS repository, can be the basis of exposing service descriptions as Linked Data. In particular, each service is managed via a unique URI, deferencable via HTTP, and linking to related datasets. The transformations presented in this paper are applied within a discovery-enabled repository, Discovery Cloud[5], interface-compatible with the SOA4All repository, iServe, for the storage of services in the minimal service model common to microWSMO and WSMO-Lite, but which also allows retrieval and inter-translation of WSMO and OWL-S services and goal/template-based discovery.

The paper is arranged as follows: the SSOA-RO is reviewed in Section 2; transformations from existing models into this ontology are considered in Section 3; transformations back out to the existing models are exemplified in Section 4; the extension of a Linked Data-compliant API to these transformations is considered in Section 5; finally, conclusions are drawn and further work discussed in Section 6.

[1] `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=semantic-ex`
[2] Semantic Evaluation at Large Scale: `http://www.seals-project.eu/`
[3] `http://about.seals-project.eu/downloads/category/1-`
[4] Service Oriented Architectures for All: `http://www.soa4all.eu/`
[5] `http://km.aifb.kit.edu/services/DisCloud`

2 OASIS Semantic SOA Reference Ontology

The Semantic SOA Reference Ontology (RO), currently under public review on the route to OASIS standardisation, is an ontology expressed in RDFS that aims to combine the features of OWL-S [4], WSMO [5] and WSMO-Lite [6], and be consistent with the terminology and concepts of the OASIS SOA Reference Model (SOA-RM) [7]. To this end the starting point for the Reference Ontology is the top-level concepts of the SOA-RM shown in Figure 1. 'Execution Context' and 'Contract & Policy' are shown dotted as the work on their semantic description is considered too early for standardisation.

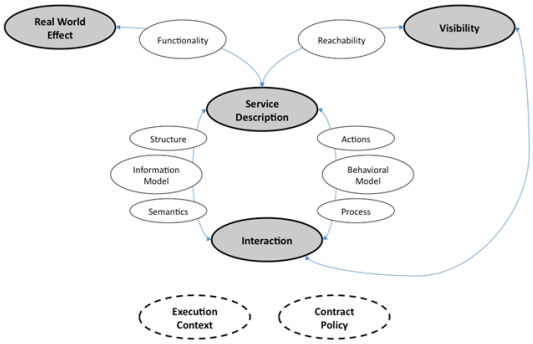


Fig. 1. Top-Level Concepts of OASIS SOA Reference Model

Figure 2 shows how these concepts are refactored and supplemented in the Semantic SOA Reference Ontology.

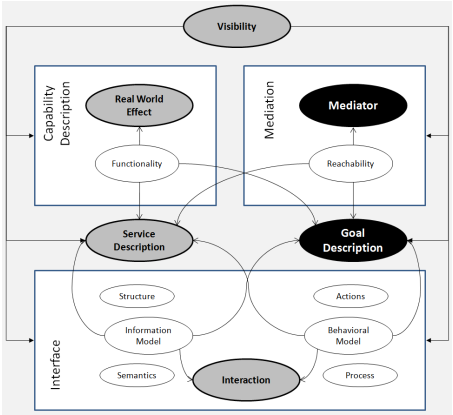


Fig. 2. Top-Level Concepts of OASIS Semantic SOA Reference Ontology

In particular, the concept of 'Visibilty' becomes a global underlying concern via the expression of the whole service model, not just the information model, in ontology-based semantics. The associated concept of 'Reachability' is subsumed into the WSMO-inspired notion of 'Mediation' (shown in black as a new top-level concept), by which top-level concepts can be connected together with a specification of the means to overcome any heterogeneities. Also novel is the WSMO-inspired concept of 'Goal', by which client requirements of a service interaction are documented. The concepts of 'CapabilityDescription' and 'Interface' are shown as groupings.

We consider now the RDFS definitions for these parts of the SSOA-RO model. The relevant 'top-level' RDFS descriptions are reproduced in Figure 3.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix ro: <http://docs.oasis-open.org/semanticsoa/referenceontology/v1.1#> .


ro:TopLevelElement rdf:type rdfs:Class .

ro:ServiceDescription rdfs:subClassOf ro:TopLevelElement .

ro:GoalDescription rdfs:subClassOf ro:TopLevelElement .

ro:Ontology rdfs:subClassOf ro:TopLevelElement ;
            rdfs:subClassOf owl:Ontology .

ro:importsOntology rdf:type rdf:Property ;
                   rdfs:domain ro:TopLevelElement ;
                   rdfs:range ro:Ontology .

ro:usesMediator rdf:type rdf:Property ;
                rdfs:domain ro:TopLevelElement ;
                rdfs:range ro:Mediator .

ro:Mediator rdfs:subClassOf ro:TopLevelElement .

ro:hasSource rdf:type rdf:Property ;
             rdfs:domain ro:Mediator ;
             rdfs:range ro:TopLevelElement .

ro:hasTarget rdf:type rdf:Property ;
             rdfs:domain ro:Mediator ;
             rdfs:range ro:TopLevelElement .
```

Fig. 3. Semantic SOA Reference Ontology Top-Level Elements in RDF(S)

Definitions, from the Reference Ontology, relevant for the 'heavyweight' definition of service capabilities, and the functional requirements of clients via goals, are shown in Figure 4. It should be noted that the Reference Ontology, and SEALS in turn, makes no particular stipulation about which language should be used to encode rules, using instead a subclass of RDF literal. When RIF[6] [8] becomes a standard it might be possible to rely on this as a common interchange for rules.

[6] http://www.w3.org/2005/rules/wiki/RIF_Working_Group

```
ro:offersCapability rdf:type rdf:Property ;
                    rdfs:domain ro:ServiceDescription ;
                    rdfs:range ro:Capability .

ro:requiresCapability rdf:type rdf:Property ;
                      rdfs:domain ro:GoalDescription ;
                      rdfs:range ro:Capability .

ro:Capability rdfs:subClassOf ro:FunctionalDescription .

ro:hasPrecondition rdf:type rdf:Property ;
                   rdfs:domain ro:FunctionalDescription ;
                   rdfs:range ro:RuleLiteral .

ro:hasAssumption rdf:type rdf:Property ;
                 rdfs:domain ro:FunctionalDescription ;
                 rdfs:range ro:RuleLiteral .

ro:hasSharedVariable rdf:type rdf:Property ;
                 rdfs:domain ro:FunctionalDescription ;
                 rdfs:range rdfs:RuleVariableLiteral .

ro:hasPostcondition rdf:type rdf:Property ;
                    rdfs:domain ro:FunctionalDescription ;
                    rdfs:range ro:RuleLiteral .

ro:hasEffect rdf:type rdf:Property ;
             rdfs:domain ro:FunctionalDescription ;
             rdfs:range ro:RuleLiteral .

ro:RuleLiteral rdfs:subClassOf rdfs:Literal .

ro:RuleVariableLiteral rdfs:subClassOf rdfs:Literal .
```

Fig. 4. Service and Goal Capability Definitions from Reference Ontology

Ranking and lightweight service discovery can be based on the properties shown in Figure 5. As well as these functional and non-functional descriptions of services, another important part of Semantic Web Service models concerns the interfaces of services. The Reference Ontology's high-level definitions for these are shown in Figure 6.

```
ro:hasNonFunctionalParameter rdf:type rdf:Property ;
                             rdfs:domain ro:TopLevelElement ;
                             rdfs:range ro:NonFunctionalParameter .

ro:requiresClassification rdf:type rdf:Property ;
                          rdfs:domain ro:GoalDescription ;
                          rdfs:range rdfs:Class .

ro:hasClassification rdf:type rdf:Property ;
                     rdfs:domain ro:ServiceDescription ;
                     rdfs:range rdfs:Class .

ro:ClassificationRoot rdfs:subClassOf rdfs:Class .
```

Fig. 5. Further (Non)Functional Definitions from Reference Ontology

```
ro:supportsInterface rdf:type rdf:Property ;
                     rdfs:domain ro:ServiceDescription ;
                     rdfs:range ro:Interface .

ro:requiresInterface rdf:type rdf:Property ;
                     rdfs:domain ro:GoalDescription ;
                     rdfs:range ro:Interface .

ro:Choreography rdfs:subClassOf ro:Interface .

ro:Orchestration rdfs:subClassOf ro:Interface .

ro:hasGlobalActionModel rdf:type rdf:Property ;
                        rdfs:domain ro:Interface ;
                        rdfs:range ro:ActionModel .
```

Fig. 6. Interface Definitions from Reference Ontology

The Reference Ontology also defines a property for Process Models for interfaces, but no standard has yet been defined for this (this is a topic for future consideration in the Technical Committee):

```
ro:hasProcessModel rdf:type rdf:Property ;
                   rdfs:domain ro:Interface ;
                   rdfs:range ro:ProcessModel .
```

Since the transformations detailed in this paper do not currently include orchestration, since this is a highly heterogeneous part of existing semantic service models, and furthermore consider only atomic processes (from a client perspective), it is sufficient to reproduce only the definitions for (choreography) Action Models, shown in Figure 7.

```
ro:hasInputAction rdf:type rdf:Property ;
                  rdfs:domain ro:ActionModel ;
                  rdfs:range ro:Action .

ro:hasOutputAction rdf:type rdf:Property ;
                   rdfs:domain ro:ActionModel ;
                   rdfs:range ro:Action .

ro:hasSharedAction rdf:type rdf:Property ;
                   rdfs:domain ro:ActionModel ;
                   rdfs:range ro:Action .

ro:communicatesConcept rdf:type rdf:Property ;
                               rdfs:domain ro:Action ;
                               rdfs:range rdfs:Class .

ro:communicatesMessage rdf:type rdf:Property ;
                               rdfs:domain ro:Action ;
                               rdfs:range rdfs:Resource .
```

Fig. 7. Action Model from Reference Ontology

3 Transformations to the Reference Ontology

In order to produce instances of the Reference Ontology from existing service collections, some form of transformation is necessary. For the most part it has been established that SPARQL is sufficient to achieve the structural transformations. A set of CONSTRUCT queries are provided for transformation into — and, as described in Section 4, out of — the Reference Ontology from the other models considered. These all rely on the following prefixes:

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
PREFIX oservice:<http://www.daml.org/services/owl-s/1.1/Service.owl#>
PREFIX oprofile:<http://www.daml.org/services/owl-s/1.1/Profile.owl#>
PREFIX oprocess:<http://www.daml.org/services/owl-s/1.1/Process.owl#>
PREFIX ogrounding:<http://www.daml.org/services/owl-s/1.1/Grounding.owl#>
PREFIX ro:<http://docs.oasis-open.org/semanticsoa/referenceontology/v1.1#>
PREFIX part:<http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/part.owl#>
PREFIX wsml:<http://www.wsmo.org/wsml/wsml-syntax#>
PREFIX sawsdl:<http://www.w3.org/ns/sawsdl#>
PREFIX wsl:<http://www.wsmo.org/ns/wsmo-lite#>
```

These queries are exemplified with the ‘OWLS2RO’ transform, whose body is given in Figure 8.

```
WHERE
 {{?service rdf:type oservice:Service} .
  {?service oservice:presents ?profile} .
  {?service oservice:describedBy ?process} .
  {?process rdf:type process:AtomicProcess} .
  {?service oservice:supports ?grounding} .
  {?grounding ogrounding:hasAtomicProcessGrounding ?processGrounding} .
  {{{?profile oprofile:hasInput ?input} .
     {?input oprocess:parameterType ?inputType} .
     OPTIONAL
      {{?processGrounding ogrounding:wsdlInput ?inputMessage} .
       {?inputMessage ogrounding:owlsParameter ?input} .
       {?inputMessage ogrounding:wsdlMessagePart ?inputPart} .
       OPTIONAL {?inputMessage ogrounding:xsltTransformationString ?inputTransform}}}
    UNION
    {{?profile oprofile:hasOutput ?output} .
     {?output oprocess:parameterType ?outputType} .
     OPTIONAL
      {{?processGrounding ogrounding:wsdlOutput ?outputMessage} .
       {?outputMessage ogrounding:owlsParameter ?output} .
       {?outputMessage ogrounding:wsdlMessagePart ?outputPart} .
       OPTIONAL {?outputMessage ogrounding:xsltTransformationString ?outputTransform}}}} .
  OPTIONAL {?profile oprofile:serviceClassification ?classification}}
```

Fig. 8. OWLS2RO Transformation Body in SPARQL

It should be noted that, since OWL-S sticks deliberately to the DL fragment of OWL, it is unable to give the range of input and output types as classes (as this would require a metaclass), and represents these simply as URIs.

This can be seen in the result of executing the body with the head 'SELECT *' over a knowledge base containing the 'BookNonMedicalFlight' service from the current 1.1 version of the OWL-S Test Collection[7], where rows with a binding of '?service' to:

```
<http://127.0.0.1/services/1.1/BookNonMedicalFlight_service.owls#
                                        BookNonMedicalFlightService>
```

include one with a binding of '?input' to:

```
<http://127.0.0.1/services/1.1/BookNonMedicalFlight_service.owls#
                                       BookNonMedicalFlight_Account>
```

and '?inputType' to:

```
"http://127.0.0.1/ontology/NonMedicalFlightCompanyOntology.owl#
                  Account"^^<http://www.w3.org/2001/XMLSchema\#anyURI>
```

In post-processing the construct query, therefore, we make direct reference (since there is no problem doing so in RDFS, or in other target languages such as WSML) to the class, but preserve the URI-based reference too, in order to recover the OWL-S description. Figure 9 shows the simple query head where just the Reference Ontology triples are created.

```
CONSTRUCT
 {?service rdf:type ro:ServiceDescription .
  ?process rdf:type ro:Choreography .
  ?process ro:hasGlobalActionModel ?processGrounding .
  ?processGrounding ro:hasInputAction ?input .
  ?input ro:communicatesConcept ?inputType .
  ?input ro:communicatesMessage ?inputMessage .
  ?inputMessage part:hasPart_directly ?inputPart .
  ?inputMessage sawsdl:loweringSchema ?inputTransform .
  ?processGrounding ro:hasOutputAction ?output .
  ?output ro:communicatesConcept ?outputType .
  ?output ro:communicatesMessage ?outputMessage .
  ?outputMessage part:hasPart_directly ?outputPart .
  ?outputMessage sawsdl:liftingSchema ?outputTransform .
  ?service ro:hasClassification ?classification .
  ?service rdfs:subClassOf ro:ClassificationRoot}
```

Fig. 9. OWLS2RO Transformation Head in SPARQL

Note that the Reference Ontology follows WSMO in grounding a class to an entire WSDL message, whereas OWL-S uses the specific WSDL 1.1 mechanism of 'parts' to map individual inputs. The transform therefore creates such a class using the W3C simple partonomy vocabulary to form the links to individual concepts. As well as replacing URIs with class references in the 'communicatesConcept' properties, the post-processing will concatenate a set of membership tests across the input types to form a WSML precondition, and form a WSML postcondition by doing the same for output types (such conditions and effects are implicit in OWL-S). Where other pre- and post-conditions exist in WSML, RIF will be used, when this becomes a standard, to express these in the most generic way possible. This is unlikely to be possible with SWRL conditions in OWL-S descriptions, but these are very scarce in practice.

[7] OWLS-TC3/htdocs/services/1.1/BookNonMedicalFlight_service.owls

Finally note that the same problem of references to classes complicates the use of functional classification in OWL-S, which is further compounded by the way in which service descriptions express the dependency on ontologies. The transform makes the associated concept a subclass of the functional classification root in the Reference Ontology, which should be propagated up the subsumption hierarchy, but this is impossible for OWL-S (though perfectly possible in WSMO and WSMO-Lite descriptions).

4 Transformation from the Reference Ontology

In order to exemplify the transformations from the SSOA-RO into existing service models, the example from the OWL-S Test Collection considered in the previous section (i.e., the OWL-S v1.1 BookNonMedicalFlight service) will be transformed to WSMO-Lite. We elide the common features and show a fragment of the RO2WSMOLite transform, primarily to show the result of the postprocessing described above, in Figure 10.

```
CONSTRUCT
  {?service rdf:type wsl:Service .
   ?precondition rdf:type wsl:Condition .
   ?precondition rdf:value ?preconditionValue .
   ?service sawsdl:modelReference ?precondition .
   ?precondition rdf:value ?preconditionValue .
   ?postcondition rdf:type wsl:Condition .
   ?postcondition rdf:value ?postconditionValue .
   ?service sawsdl:modelReference ?postcondition .
   ?postcondition rdf:value ?postconditionValue .
}
WHERE
 {?service rdf:type ro:ServiceDescription .
  ?service ro:hasCapability ?capability .
  ?capability ro:hasPrecondition ?precondition .
  ?precondition rdf:value ?preconditionValue .
  ?capability ro:hasPostcondition ?postcondition .
  ?postcondition rdf:value ?postconditionValue}
```

Fig. 10. OWLS2RO Transformation Head in SPARQL

The result of applying this query to the post-processed version of the example considered in the previous section is shown, as a screenshot of the execution of the query in the Sesame Workbench, in Figure 11.

In many cases the SAWSDL for service descriptions has already been produced, though may be updated to include references such as those shown to new WSML expressions. Where they do not exist they will be created by extension of the existing WSDL. In each case these will be stored, alongside the RDF representation in the Reference Ontology in the Test Data Repository.

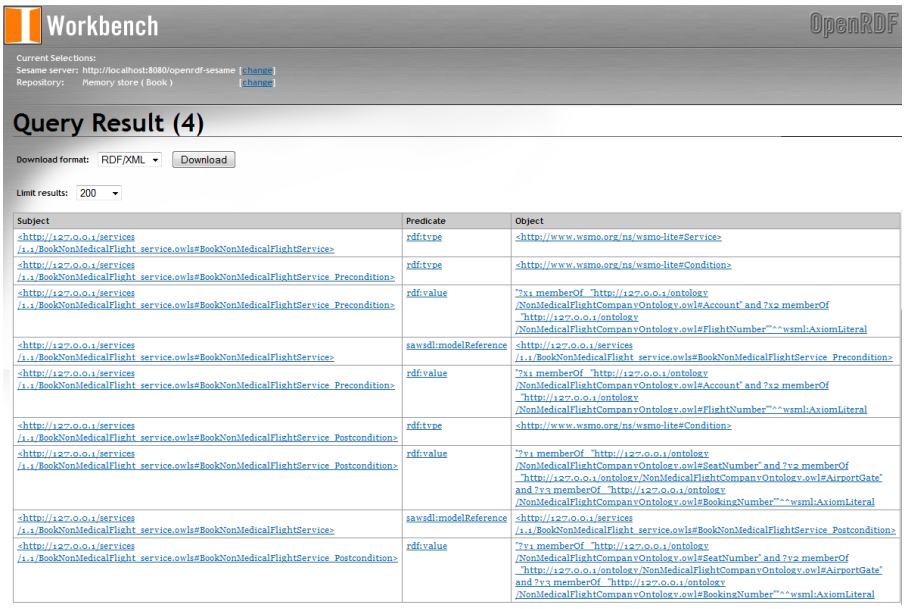


Fig. 11. Transformation of OWL-S TC 1.1 Flight Booking Service to WSMO-Lite

5 Service Descriptions as Linked Data

The four so-called 'Linked Data Principles' are originally stated[8] as follows:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
4. Include links to other URIs. so that they can discover more things.

In a resource-oriented, i.e, truly RESTful, service interface it is natural that service descriptions are identified by resolvable URIs (principles 1 and 2), and this was the approach taken for repositories in the SEALS project and later SOA4All. Furthermore, as has been shown, all service models can be represented in RDF (principle 3) and this is the approach taken in the repository interface in iServe and here. Finally services link to ontologies used in their description, in particular domain ontology import is exposed as a subproperty to RDF's 'seeAlso' (principle 4 — further means to meet this principle are considered in Section 6.

[8] `http://www.w3.org/DesignIssues/LinkedData.html`

In order to respect Linked Data and REST principles, and remain compatible with the iServe API used in the SOA4All project, the extensions described in this paper are encoded as an extension to content negotiation. In HTTP, URIs identify *resources* but these might have different *representations* that can be retrieved. A common example is in the format used for a picture — a client might ask (preferentially) for a JPG over a GIF encoding. Similarly in Linked Data we might allow retrieval of RDF content in RDF/XML, Turtle (n3) etc. The repository interface supported here allows exactly specification of the RDF format required in the usual 'accept' field to the request header.

The service model in which the result should be returned, however, is orthogonal to the RDF format. One might request a WSMO description in N3, or an OWL-S description in RDF/XML, for instance. For this reason we introduce a second header field, which acts like a simplified version of accept headers and negotation, with the key 'service_model' and values that are URIs for each of the service models.

In both cases, i.e. the service model and the RDF encoding, furthermore, there should be some default in case the client makes no specific request. In order to remain compatible with iServe the default service model is the 'minimal service model' of WSMO-Lite/microWSMO. Since it is the Web representation, the default RDF encoding is RDF/XML.

6 Conclusions and Further Work

This paper has detailed the Semantic SOA Reference Ontology, its definition in RDFS, and its utility in transforming between existing service models to increase interoperability in the usage of semantic repositories for service descriptions. Concretely it has shown how SPARQL is, for the most part, sufficient to achieve these transformations. Finally it has been shown how an API, compatible with Linked Data principles, can be formed to expose these transformations over a service repository in a RESTful fashion.

On-going work on the repository considers the treatment of templates, used as the basis for discovery, as permanent resources, i.e. uploaded to the repository just as service descriptions, where discovery is carried out on an on-going basis as new service descriptions are found by crawling and/or uploaded. A 'GET' retrieval on the service template can therefore also return a dynamically-ranked set of services that can be used to achieve the template behaviour, providing further justification for the claims with respect to the fourth Linked Data principle.

Future work on the Reference Ontology will consider process models sufficient for orchestration and it is hoped that these will be derivable from the OWL-S process model, and that restricted processes will be capable of transformation into the OWL-S process model, WSML Abstract State Machine-based orchestrations and semantic BPEL extensions, as already considered — with transformations based on semantic rule languages — in [9].

Acknowledgements: The work is supported by the EU FP7 ICT projects SOA4All (IP 215219) and SEALS (e-Infrastructures 238975). We gratefully acknowledge the insights of project participants Barry Bishop, Reto Krummenacher and Carlos Pedrinaci, as well as the contributions of all members of the OASIS SEE Technical Committee.

References

1. Haller, A., Cimpian, E., Mocan, A., Oren, E., Bussler, C.: WSMX - A Semantic Service-Oriented Architecture. In: Proceedings of the International Conference on Web Services (ICWS2005), Orlando, Florida, USA (July 2005)
2. Cabral, L., Domingue, J., Galizia, S., Gugliotta, A., Norton, B., Tanasescu, V., Pedrinaci, C.: IRS-III: A broker for semantic web services based applications. In: Proceedings of the 5th International Semantic Web Conference (ISWC2006), Athens, Georgia, USA (Nov 2006)
3. Verma, K., Gomadam, K., Sheth, A., Miller, J., Wu, Z.: The METEOR-S Approach for Configuring and Executing Dynamic Web Processes. Technical report, LSDIS (24 June, 2005)
4. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S: Semantic markup for web services. Available at `http://www.daml.org/services/owl-s/1.1/overview/` (November 2004)
5. Fensel, D., Lausen, H., Polleres, A., de Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services. Springer (2006)
6. Vitvar, T., Kopecky, J., Viskova, J., Fensel, D.: Wsmo-lite annotations for web services. In Hauswirth, M., Koubarakis, M., Bechhofer, S., eds.: Proceedings of the 5th European Semantic Web Conference. Number 5021 in LNCS, Springer Verlag (2008)
7. OASIS SOA Reference Model TC: Reference model for service oriented architecture 1.0. Technical report, OASIS (October 2006)
8. Kifer, M.: Rule interchange format: The framework. In: RR '08: Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems. Number 5341 in LNCS, Springer (2008) 1–11
9. Norton, B., Cabral, L., Nitzsche, J.: Ontology-based translation of business process models. In: Proceedings of 4th International Conference on Internet and Web Applications and Services (ICIW 2009), IEEE Computer Society (2009) 481–486

iServe: a Linked Services Publishing Platform

Carlos Pedrinaci, Dong Liu, Maria Maleshkova, David Lambert, Jacek Kopecký, and John Domingue

Knowledge Media Institute, The Open University
Walton Hall, Milton Keynes, MK7 6AA, UK
c.pedrinaci@open.ac.uk

Abstract. Despite the potential of service-orientation and the efforts devoted so far, we are still to witness a significant uptake of service technologies outside of enterprise environments. A core reason for this limited uptake is the lack of appropriate publishing platforms able to deal with the existing heterogeneity in the service technologies landscape and able to provide expressive yet simple and efficient discovery mechanisms. In this paper we describe iServe, a novel and open platform for publishing services which aims to better support their discovery and use. It exposes service descriptions as linked data expressed in terms of a simple vocabulary for describing services of different kinds with annotations in diverse formalisms. In addition to describing iServe, this paper also highlights the set of principles behind iServe, which we believe are essential for other generic repositories of semantic information notably ontology repositories.

1 Introduction

Web services are software systems offered over the Internet via platform and programming-language independent interfaces defined on the basis of a set of open standards such as WSDL, SOAP, and further WS-* specifications [1]. The fundamental advantage of Web service technology lies in the support it brings to developing highly complex distributed systems that maximise reuse of loosely coupled components. A key constituent of Service-Oriented Architectures is the service repository, which enables programmatic recording of service descriptions and their subsequent use in the discovery of suitable services. Service publication has therefore been at the core of research and development in this area since the very beginning. However, despite substantial efforts, Web services are not published on the Web in significant numbers, and in practice, lighter and less structured approaches such as Web APIs are currently preferred [2].

One of the main reasons for the paucity of service repositories to date has been the fact that, although they are relatively complex, they do not support expressive queries, limiting their usefulness [3]. Semantic Web Services (SWS) [4] research has devoted considerable efforts to overcoming Web services limitations by enriching them with semantic annotations to better support their discovery, composition and execution. So far, the impact of SWS on the Web has been minimal. In fact, although SWS technologies have already demonstrated benefits,

research in the area has glossed over the additional effort demanded of users, and the extra complexity they introduce to the already intricate services technology stack.

Before any significant uptake of services can take place on the Web, better mechanisms for creating, publishing and discovering services must be in place. In particular, service publication must be able to deal with service heterogeneity (e.g., dealing with both WSDL services and Web APIs), it must be based on the use of lightweight semantics able to support relatively advanced yet efficient discovery, and it must be combined with an appropriate set of tools able to support users in the annotation and publication of services.

In this paper we describe iServe, a platform for the seamless publication and discovery of services developed in the context of the EU project SOA4All. iServe addresses the publication of services from a novel perspective based on lessons learned from the evolution of the Web of Data [5]. iServe transforms service annotations expressed in a variety of formats into what we refer to as Linked Services—linked data describing services—that can directly be interpreted by state of the art Semantic Web technologies for their discovery and further processing. The iServe platform is complemented with editors that assist users in creating and publishing service annotations using existing semantic search engines like Watson [6] for searching the Web for reusable ontologies. The decisions adopted for iServe include a number of principles that are of particular relevance for the development of other kinds of repositories for the Web, since they highlight the importance that ontology repositories or systems like Watson may have when it comes to creating Semantic Web applications.

The remainder of this paper is organised as follows. First we cover the state of the art in services description and publication (Section 2). We then introduce iServe and the core principles that underpin the approach (Section 3). Next, we discuss openness as one of the essential characteristics of the approach, and highlight how existing editors and ontology indexing systems can be connected to iServe to better support the creation and publication of service annotations (Section 4). Finally, we present our main conclusions and introduce lines for future research (Section 5).

2 Background and Related Work

The Universal Business Registry part of Universal Description Discovery and Integration (UDDI) [7] is perhaps the best-known effort to support the publication of services on the Web. On the basis of UDDI, large companies like SAP, IBM and Microsoft created a universal registry for enterprise services that could be accessed publicly but it did not gain enough adoption and it was discontinued in 2006 after five years of use. Today, Seekda![1] provides one of the main repositories of publicly available Web services. Seekda's repository currently lists 28,500 Web services with their corresponding documentation, and this number seems to be

[1] See http://webservices.seekda.com/

stagnant. The number of services publicly available contrasts significantly with the billions of Web pages available, and interestingly is not significantly bigger than the 1,500 services estimated to be deployed internally within Verizon [8]. Other academic efforts in crawling and indexing of Web services on the Web have found far lower numbers of services [9].

A major reason for the lack of success of repositories such as UDDI was the fact that, although these registries are relatively complex, they do not support expressive queries, limiting their usefulness [3]. As a consequence, developers did not use these systems since the benefits were often not worth the extra effort. A second reason for the lack of uptake is that Web services have so far essentially targeted enterprises, which as we saw earlier, do not publish services in any significant numbers.

The Web services ecology has recently seen a major evolution with the advent and proliferation of Web APIs and RESTful services [10], and there has not been much progress on, or even concern with, means for describing and discovering these newer kinds of services. Perhaps the most popular directory is ProgrammableWeb.com, which at the time of this writing lists 1,700 APIs and 4,600 mashups. This directory is based on the manual submission of APIs by users and currently provides simple search mechanisms based on keywords, tags, or a simple classification, none of which are particularly expressive. APIHut [11] is a platform that claims to increase the accuracy of keyword-based search of APIs compared to ProgrammableWeb.com or plain Google search, although it does not provide richer discovery mechanisms.

SWS [4] have long tried to overcome the limitations of Web service descriptions by enriching them with semantic annotations. The landscape of SWS is characterised by a number of conceptual models that, despite a few common characteristics, remain essentially incompatible due to the different representation languages and expressivity utilised, as well as because of conceptual differences. Major frameworks include WSMO [12], OWL-S [13], SAWSDL [14], and WSMO-Lite [15]. Regardless of the differences at the semantic level, the vast majority of the SWS initiatives are predicated upon the semantic enrichment of WSDL Web services, and these have turned out not to be prevalent on the Web. Only recently have researchers started focusing on Web APIs and RESTful services, the main examples being SA-REST [16] and MicroWSMO [17].

Enhancing service repositories with semantics and supporting automated discovery has been one of the key topics SWS research has tackled [18–20]. Despite demonstrating advantages of semantic annotations in discovering services, particularly in terms of accuracy and in dealing with heterogeneous data models, SWS work has overlooked the additional complexity involved in creating semantic annotations for services. Consequently, there is no significant body of SWS published in a convenient manner on the Web: the largest public SWS repository today is probably OPOSSum, a test collection with less than 3000 service annotations which provides programmatic access to its content solely through direct access to the database management system [21].

3 Services Publication as Linked Data

The current state of the art evidences limited use of service technologies on a Web scale, and existing technologies for publishing and discovering services remain rather simple, providing limited support and usually based on keyword-based search. This type of mechanism has proven to be insufficient for the needs of software developers, hampering uptake [3]. At the other end of the spectrum, SWS research has aimed for highly advanced discovery techniques but has instead created additional overheads, notably a considerable bottleneck for the creation of rich annotations.

The Web of Data is a relatively recent effort, derived from research on the Semantic Web, whose main objective is to generate a Web exposing and interlinking data previously isolated in silos. The Web of Data is based upon four simple principles, known as the linked data principles, which essentially dictate that every piece of data should be given an HTTP URI which, when looked up, should offer useful information using standards like RDF and SPARQL [5]. Moreover, data should be linked to other relevant resources, thereby allowing humans and computers to discover additional information. Since the linked data principles were outlined in 2006 they have been widely adopted in academic environments, large companies (like the BBC), and national governments (including the United Kingdom's), all of whom are progresively publishing large amounts of data expressed in terms of lightweight ontologies often referred to as vocabularies.

iServe is a novel and open platform for publishing semantic annotations of services based on a direct application of linked data principles to publish service annotations expressed in terms of a simple vocabulary for describing services of different kinds (e.g., WSDL and Web APIs) with annotations in diverse formalisms (e.g., OWL-S, WSMO-Lite). More concretely, iServe is driven by the following conclusions drawn from previous research on service repositories and the progress made by the Web of Data:

- Semantics are essential to reach a minimum level of automation during the life-cycle of services;
- The annotation of services should be simplified in as much as possible;
- On the Web, lightweight ontologies together with the possibility to provide custom extensions prevail against more complex models;
- Any solution to deploying services that aspires to be widely adopted should build upon the various approaches and standards used on the Web, including Web APIs, RDF, and SPARQL;
- Linked data principles are an appropriate means for publishing large amounts of semantic data, both for human and machine consumption;
- Links between publicly available datasets are essential for the scalability and the value of the data exposed.

In the remainder of this section we describe iServe in detail, focusing first on the overall approach and architecture, then describing the ontology it uses, and finally presenting the import mechanisms that make it largely compatible with a wide range of service annotations.

3.1 Overall Approach

iServe supports publishing Linked Services expressed in terms of a simple conceptual model that is suitable for use by both humans and machines, and which abstracts away the existing heterogeneity around service kinds and annotation formalisms. In particular iServe:

- supports importing service annotations in a range of formalisms (e.g., SAWSDL, WSMO-Lite, MicroWSMO) that cover both WSDL services and Web APIs;
- provides means for publishing semantic annotations of services which are automatically assigned a resolvable HTTP URI;
- includes support for content negotiation so that service annotations can be returned in HTML for human users, or in RDF for machine interpretation;
- provides a SPARQL endpoint allowing advanced querying over the services annotations;
- offers a read/write REST API so that services can easily be retrieved and published from remote applications; and
- automatically generates links between the published service annotations and additional documents on the Web such as the original service description or documentation so that users and machines can easily discover more information.

The architecture of iServe, depicted in Figure 1, comprises a crawler, a RESTful API, a set of import mechanisms, and an RDF store. The crawler collects existing annotations from the Web in order to publish them in iServe. Given that not many annotations are published on the Web, the crawler currently deals only with known sets of service annotations such as the SAWSDL retrieval test collection (SAWSDL-TC)[2] and the OWL-S retrieval test collection (OWLS-TC)[3]. The RESTful API, implemented using Restlet[4], provides operations for accessing service annotations and service documentation, allowing remote applications to publish and discover services. The import mechanisms provide support for importing annotations in diverse formalisms by transforming them into the Minimal Service Model, explained in more detail below, which is used by iServe for publishing them as linked data. This service model provides a common vocabulary for service annotations, smoothing away the heterogeneity of different formalisms such as OWL-S, SAWSDL and WSMO, allowing humans and machines to discover service annotations originally described using heterogeneous conceptual models through a single vocabulary. Finally, iServe captures the service annotations together with some provenance information including the annotation author and the creation or modification date in an RDF store. RDF storage and querying support is provided by Swift OWLIM,[5] although it is accessed via RDF2Go[6] to maintain independence with respect to the concrete store used. The RDF

[2] See http://www.semwebcentral.org/projects/sawsdl-tc/
[3] See http://www.semwebcentral.org/projects/owls-tc/
[4] See http://www.restlet.org/
[5] See http://www.ontotext.com/owlim/
[6] See http://semanticweb.org/wiki/RDF2Go

store provides us with a SPARQL endpoint that is made available to external applications in order to interact with iServe for retrieving services.

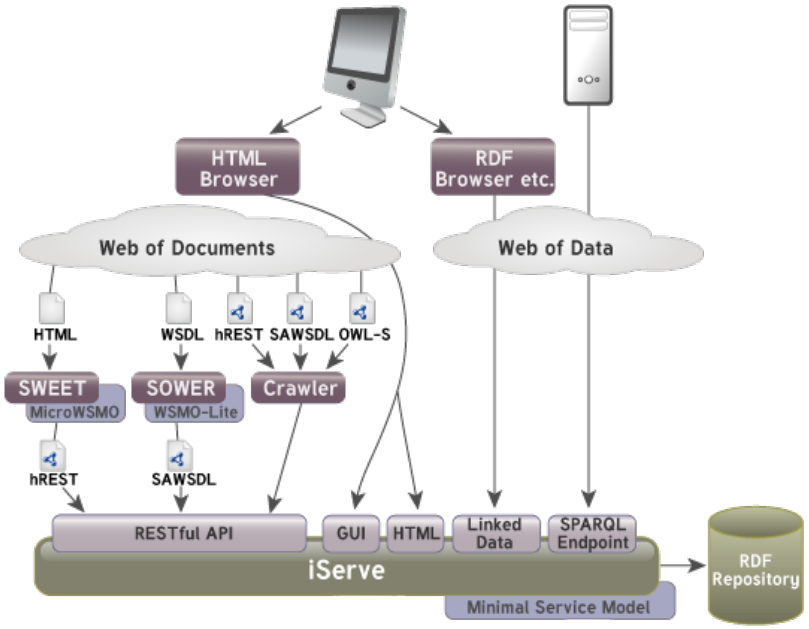


Fig. 1. The overall architecture of iServe.

3.2 Conceptual Models

Section 2 covered the state of the art around the publication of services and semantic annotations, highlighting the existing heterogeneity in terms of languages and formalisms used, the main approaches adopted thus far for publishing and discovering services, and the need of finding a trade-off between expressivity of service descriptions and their complexity. In building a system like iServe it is necessary to provide a common vocabulary, able to describe services in a way that allows machines to automatically locate and filter services according to their functionality or the data they handle, independent of the formalism originally used to describing them.

The best-known approaches to annotating services semantically are OWL-S, WSMO, SAWSDL, and WSMO-Lite for WSDL services, and MicroWSMO, and SA-REST for Web APIs. To cater for interoperability, iServe uses what

can essentially be considered the largest common denominator between these formalisms which we refer to as the Minimal Service Model (MSM). The MSM, first introduced together with WSMO-Lite [15], is a simple RDF(S) ontology able to capture the core semantics of both Web services and Web APIs in a common model supporting the common publishing and search of services, yet still permitting framework-specific extensions to remain attached and thereby benefitting those clients able to comprehend those formalisms.

The MSM, denoted by the `msm` namespace in Figure 2, defines `Service`s which have a number of `Operation`s. `Operation`s in turn have input and output `MessageContent` descriptions, and `Fault`s. A `MessageContent` may be composed of `MessagePart`s which can be mandatory or optional. The addition of message parts extends the earlier definition of the MSM as described in [15]. The intent of the message part mechanism is to support finer-grain discovery based on message parts, mirroring the granularity of SAWSDL and allowing to distinguish between mandatory and optional parts.

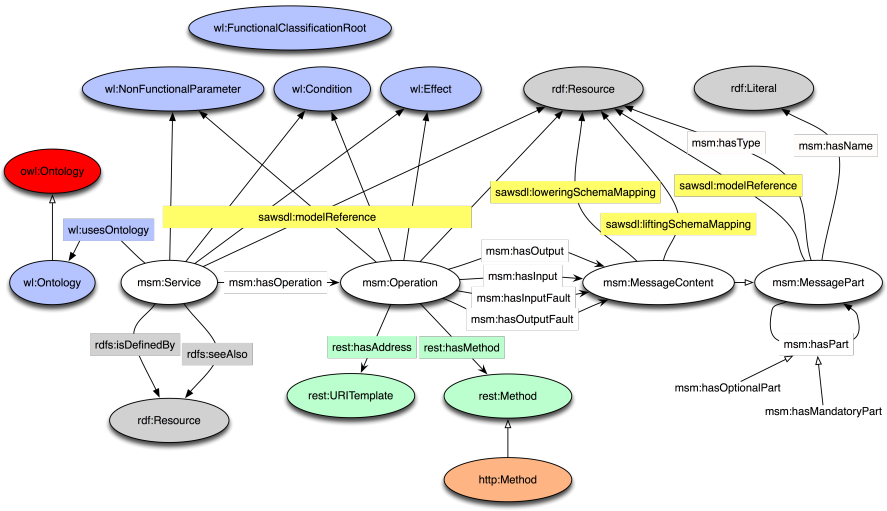


Fig. 2. Conceptual model for services used by iServe.

iServe also uses the SAWSDL, WSMO-Lite and hRESTS vocabularies, depicted in the figure with the `sawsdl`, `wl`, and `rest` namespaces respectively. The SAWSDL vocabulary captures in RDF the three kinds of annotations over WSDL and XML Schema, namely `modelReference`, `liftingSchemaMapping` and `loweringSchemaMapping` that SAWSDL supports. SAWSDL supports the annotation of WSDL and XML Schema syntactic service descriptions with semantic concepts, but does not specify a particular representation language nor does it provide any specific vocabulary that users should adopt. The `modelReference`

property links syntactic service elements to semantic models via URIs, while the schema mapping properties indicate data transformations from a syntactic representation to its semantic counterpart and vice versa.

WSMO-Lite builds upon SAWSDL by extending it with a model specifying the semantics of the particular service annotations. It provides a simple RDFS ontology together with a methodology for expressing functional and non-functional semantics, and an information model for WSDL services based on SAWSDLs modelReference hooks. In particular, WSMO-Lite makes explicit the intended meaning for `modelReference` annotations without modifying SAWSDL. Instead, it provides a vocabulary to annotate the URIs pointed to by SAWSDL annotations. The WSMO-Lite vocabulary includes the classes `NonFunctionalParameter`, `FunctionalClassificationRoot`, `Condition`, `Effect` and `Ontology`. With these, an annotator can type SAWSDL annotation references without adding information directly to the WSDL description.

The hRESTS vocabulary [22] extends the MSM with specific attributes for operations to model information particular to Web APIs, such as a `URITemplate` to describe the URI for invocation, and `method` to indicate the HTTP method used for the invocation. For methods, iServe uses the draft W3C HTTP in RDF vocabulary [23].

4 An Open Publishing Platform

The fundamental objective pursued by iServe is to provide a platform able to publish service annotations in a way that would allow people to achieve a certain level of expressivity and refinement in discovering services, while remaining simple and convenient both for human and machine use. The simple conceptual model explained earlier is a principal building block to support this as a general model able to abstract away the existing conceptual heterogeneity among service annotation approaches without introducing considerable complexity from a knowledge acquisition and computation perspectives. Thanks to its simplicity, the MSM captures the essence of services in a way that can support service matchmaking and invocation and still remains largely compatible with the RDF mapping of WSDL [24], with WSMO-based descriptions of Web services, with OWL-S services, and with services annotated according to WSMO-Lite and MicroWSMO. Although providing a formal mapping for each of these languages is out of the scope of this paper, we note that the elements captured in the MSM are largely common to existing models. The mapping is not lossless, but appropriate use of `rdfs:isDefinedBy`, covered next, can help circumvent this limitation and still provide a common ground for publishing Linked Services in the Web of Data in a way that is amenable to automated processing and where more expressive definitions can be linked if needed.

iServe provides a set of import mechanisms that can take service annotations in several formats, and generate the appropriate RDF in terms of iServe's conceptual model. In particular, the current version can import SAWSDL, WSMO-Lite, MicroWSMO, and part of OWL-S descriptions. The import process generates

`rdf:isDefinedBy` links from the service annotation to the original description file (e.g., the WSDL including the annotations), `rdf:seeAlso` to any documentation about the service (e.g., the Web API description page itself), and `owl:sameAs` relations to the published version of the same service annotation in other systems. Currently, the only `owl:sameAs` links that are generated link to the RDF mapping of WSDL [24], ensuring compatibility with tools using it, but in the future, other links could be generated to connect services stored in other repositories adopting principles similar to iServe's.

In addition to the import mechanisms that provide compatibility and abstract away the heterogeneity in service description formalisms, iServe contributes to publishing services on the Web by automatically hosting service descriptions at a publicly accessible URI, and by offering means for users or machines to upload service annotations through its RESTful API[7].

iServe is integrated with two service annotation editors being developed within the SOA4All project: SWEET [25] (SemanticWeb sErvices Editing Tool) and SOWER (SWEET is nOt a Wsdl EditoR), which support users in annotating Web APIs and WSDL services respectively.[8] Both editors are Web applications that can directly be used with a Web browser and provide support for browsing service descriptions and annotating them through a simple interface. Behind the scenes, both editors take care of serialising the descriptions according to MicroWSMO (SWEET) and WSMO-Lite (SOWER) specifications respectively. Both editors strive to assist users in creating service annotations through two main means: the use of the Web as background knowledge and a direct connection with iServe for the seamless persistence and publication of annotations.

A fundamental part of service annotation concerns the linking of (parts of) service descriptions to ontologies capturing semantically the data model they handle, certain non-functional properties such as the quality of service and price, and their categorisation with respect to reference taxonomies and classifications of services. Thus far, the authoring of service annotations has essentially been based on the manual creation of both the services' structure and the actual ontologies used in their annotation. Consequently, creating service annotations has been a particularly tedious task which rather than better supporting the integration of services has led instead to additional heterogeneity at the semantic level. Further limitations have also traditionally been brought by the fact that the service annotations and the ontologies used were most often not published publicly which impedes the interpretation and use of services by third-parties.

SWEET and SOWER support the integrated search of suitable domain ontologies by means of Watson [6]. Watson serves as a gateway for the Semantic Web by collecting the available semantic content on the Web, analyzing it to extract useful metadata and generate indexes, and implementing efficient query facilities to access the semantic data. During the annotation of services, Watson supports retrieving ontologies and concepts matching particular keywords. In this way, a user can select a service property and receive a list of semantic entities

[7] More details about the API can be found at http://iserve.kmi.open.ac.uk
[8] A publically accessible SWEET instance is available at http://sweet.kmi.open.ac.uk.

suitable for the annotation thus potentially reducing the manual labour involved through reuse (Figure 3). Integrating the editors with systems like Watson gives annotators better access to semantic vocabulary, hopefully leading to descriptions that are both more precise, and more widely understood by dint of being found on the Web and reused rather than invented on an ad-hoc basis, thereby embedding service descriptions in the existing Web of linked data.

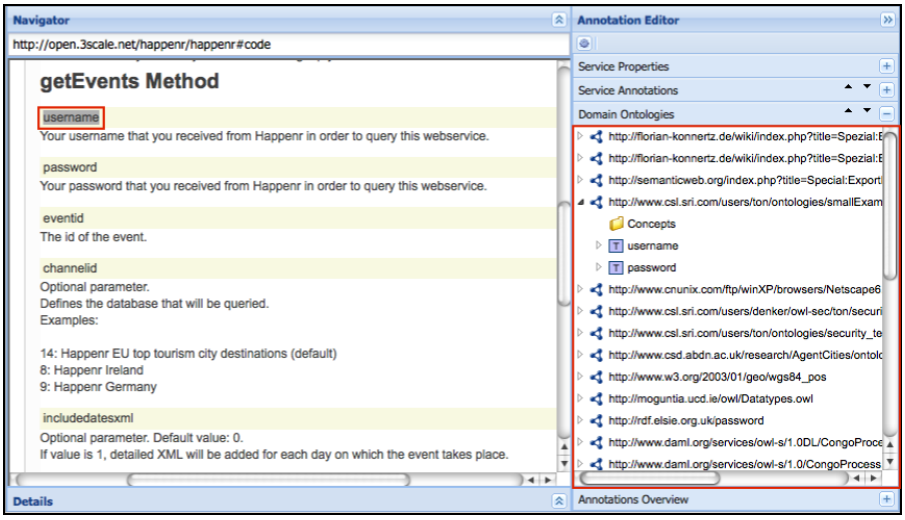


Fig. 3. SWEET showing possible annotations based on Watson results.

5 Conclusions and Future Work

Despite the potential of service technologies and the efforts devoted so far, we have yet to witness a significant uptake of service technologies outside enterprise environments. There is a tension between the rich descriptions necessary to automate much of the use of services, and the reticence on the part of developers to invest that effort. A similar situation exists in Semantic Web Services, where the expressive models able to capture the semantics of services stand in contrast to the lightweight tagging approaches that appear more acceptable to users.

In this paper we described iServe, a novel and open platform for publishing services that aims to better support their discovery and use by exposing them as linked data expressed in terms of a simple vocabulary for services. This vocabulary provides a common ground for descriptions of different flavours of Web services (SOAP services described in WSDL, and RESTful services described in hRESTS), and several kinds of semantic formalisms (including OWL-S, WSMO, and WSMO-Lite). iServe is underpinned by principles we believe are applicable

to generic repositories of semantic information. These principles include the application of linked data principles to publish metadata about the content, the automated publication of the actual content at a public URI if necessary, the adoption of a simple vocabulary able to abstract away the heterogeneous vocabularies or formalisms used by other repositories within the domain, and the provisioning of an open API allowing applications to automatically retrieve and publish information.

Additionally in this paper we have shown how the advent of engines like Watson that support searching for ontological entities over the Web has enabled a new way for authoring semantic models which as we have seen can also contribute to authoring service annotations by reducing the amount of effort required and potentially contributing to reducing integration issues through ontology reuse.

Future work on iServe will focus on the development and refinement of import mechanisms for WSMO and OWL-S, the creation of advanced indexing mechanisms, and capturing non-functional information about services gleaned at runtime from the monitoring infrastructure. Future work on the editor side will focus on better assisting users in the creation of annotations.

Acknowledgements This work was partly funded by the EU project SOA4All (FP7-215219). We thank all the members of the SOA4All project and the Conceptual Models for Services Working Group of STI International. We also thank Pierre Grenon for his insightful comments and Alex Simov for the development of SOWER.

References

1. Erl, T.: SOA Principles of Service Design. The Prentice Hall Service-Oriented Computing Series. Prentice Hall (July 2007)
2. Davies, J., Domingue, J., Pedrinaci, C., Fensel, D., Gonzalez-Cabero, R., Potter, M., Richardson, M., Stincic, S.: Towards the open service web. BT Technology Journal **26**(2) (2009)
3. Pilioura, T., Tsalgatidou, A.: Unified publication and discovery of semantic web services. ACM Trans. Web **3**(3) (2009) 1–44
4. McIlraith, S., Son, T., Zeng, H.: Semantic web services. IEEE Intelligent Systems **16**(2) (March 2001) 46–53
5. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. International Journal on Semantic Web and Information Systems (IJSWIS) (2009)
6. d'Aquin, M., Motta, E., Sabou, M., Angeletou, S., Gridinoc, L., Lopez, V., Guidi, D.: Toward a new generation of semantic web applications. IEEE Intelligent Systems **23**(3) (2008) 20–28
7. Clement, L., Hately, A., von Riegen T. Rogers, C.: UDDI Specification Version 3.0.2. Technical report, OASIS (2004)
8. Stollberg, M.: Scalable Semantic Web Service Discovery for Goal-driven Service-Oriented Architectures. PhD thesis, Faculty of Mathematics, Computer Science and Physics Leopold-Franzens University Innsbruck, Austria (March 2008)
9. Al-Masri, E., Mahmoud, Q.H.: Investigating web services on the world wide web. In: WWW '08: Proceeding of the 17th international conference on World Wide Web, New York, NY, USA, ACM (2008) 795–804

10. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly Media, Inc. (May 2007)
11. Gomadam, K., Ranabahu, A., Nagarajan, M., Sheth, A.P., Verma, K.: A faceted classification based approach to search and rank Web APIs. In: ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services, Washington, DC, USA, IEEE Computer Society (2008) 177–184
12. Fensel, D., Lausen, H., Polleres, A., de Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services: The Web Service Modeling Ontology. Springer (2007)
13. Martin, D., Burstein, M., J., H., Lassila, O., McDermott, D., McIlraith, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/owl-s/1.0/owl-s.pdf (2004)
14. Farrell, J., Lausen, H.: Semantic Annotations for WSDL and XML Schema. http://www.w3.org/TR/sawsdl/ (January 2007) W3C Candidate Recommendation 26 January 2007.
15. Vitvar, T., Kopecky, J., Viskova, J., Fensel, D.: WSMO-Lite annotations for web services. In Hauswirth, M., Koubarakis, M., Bechhofer, S., eds.: Proceedings of the 5th European Semantic Web Conference. LNCS, Berlin, Heidelberg, Springer Verlag (June 2008)
16. Sheth, A.P., Gomadam, K., Lathem, J.: SA-REST: Semantically interoperable and easier-to-use services and mashups. IEEE Internet Computing **11**(6) (2007) 91–94
17. Maleshkova, M., Kopecký, J., Pedrinaci, C.: Adapting SAWSDL for semantic annotations of restful services. In: Workshop: Beyond SAWSDL at OnTheMove Federated Conferences & Workshops. (2009)
18. Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated discovery, interaction and composition of semantic web services. Web Semantics: Science, Services and Agents on the World Wide Web **1**(1) (2003) 27 – 46
19. Klusch, M., Fries, B., Sycara, K.: Automated semantic web service discovery with OWLS-MX. In: AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM (2006) 915–922
20. Stollberg, M., Hepp, M., Hoffmann, J.: A caching mechanism for semantic web service discovery. In: 6th International and 2nd Asian Semantic Web Conference (ISWC2007+ASWC2007). (November 2007) 477–490
21. Küster, U., König-Ries, B.: Towards standard test collections for the empirical evaluation of semantic web service approaches. Int. J. Semantic Computing **2**(3) (2008) 381–402
22. Kopecky, J., Gomadam, K., Vitvar, T.: hRESTS: an HTML Microformat for Describing RESTful Web Services. In: The 2008 IEEE/WIC/ACM International Conference on Web Intelligence (WI2008), Sydney, Australia, IEEE CS Press (November 2008)
23. Koch, J., Velasco, C.A.: HTTP vocabulary in RDF 1.0. Working draft, W3C (October 2009)
24. Kopecký, J.: Web services description language (WSDL) version 2.0: RDF mapping. Working group note, W3C (June 2007)
25. Maleshkova, M., Pedrinaci, C., Domingue, J.: Supporting the creation of semantic RESTful service descriptions. In: Workshop: Service Matchmaking and Resource Retrieval in the Semantic Web (SMR2) at 8th International Semantic Web Conference. (2009)

Context-aware access to ontologies on the Web

Patrick Maué[1], Alejandro Llaves Arellano[1], and Thore Fechner[1]

Institute for Geoinformatics (ifgi), University of Muenster, Germany
patrick.maue|alejandro.llaves|thore.fechner@uni-muenster.de

Abstract. Domain vocabularies capture the ontology engineer's context-specific perspective on reality. Existing solutions for serving such ontologies often lack intuitive means to avoid conflicts due to logically inconsistent concept descriptions. In addition, no efficient and simple techniques for selecting only relevant terms from extensive vocabularies exist. We present an implementation of a concept repository which shifts the focus from ontologies towards individual concept descriptions. The description's identity is defined by its title and an optional set of subjects. We introduce the notion of profiling concept descriptions to distinguish between context-independent and context-specific (and potentially conflicting) properties. A flexible approach for constructing the concept identifiers supports context-aware access. Furthermore, an extensible set of query actions allows for retrieving certain parts of ontologies, e.g. the neighbourhood of one particular concept or all concepts associated with a certain subject. We illustrate the findings with an implementation of an ontology repository.

1 Introduction

Integrating information across domains relies on a consistent interpretation of the underlying data models. Such semantic interoperability depends on mappings between different domain-specific vocabularies[1]. Ontologies are commonly used to formally represent such vocabularies. Aligning these ontologies to upper-level ontologies , or creating rules mapping between potentially conflicting domain ontologies [2], ensures integration without losing domain-characteristic features. Reasoning engines use the alignments to infer matching conceptualisations. This long-term vision of semantic interoperability across information communities is based, amongst others, on the assumption that:

(a) all domain ontologies are published on the Web. The ontology elements are resources with an identity, and are *accessible* [3] via unique and resolvable identifiers. Such Uniform Resource Locators (URL) are required for relating local application-specific schema to terms in shared vocabularies, and let reasoners retrieve the descriptions from the Web [4].
(b) the relationships between ontology elements are consistent and valid. URLs used by relations referring to external terms have to be accessible and return a valid resource.

(c) elements in local application schema are referenced to shared vocabularies using semantic annotations [5].

Ontologies are traditionally implemented as downloadable files encoded in one particular ontology language. Scope and encoding are defined by the ontology engineer, the ontology's content is usually static. Even though this approach complies to the assumptions (a) and, if performed carefully, also (b), it poses a great problem for (c). Only few examples of accessible and actively re-used ontologies exist. These are usually abstract and thematically narrow proposals such as FOAF [6] for modelling social networks. Published domain-specific ontologies, e.g. the SWEET ontologies for Earth science [7], are either only partially re-used or heavily adapted to local needs. Reasons for this are, amongst others: they are too extensive, which impairs navigation and understanding. They are too limited in scope. They are biased and don't capture the shared consensus of different domain experts. They are not maintained and therefore not representing the current state of knowledge. Or they are inconsistent, linking to remote, but non-existent resources. The implementation of the concept repository (CORE) addresses the first three issues, with the potential to also target the last two. An in-depth discussion of basic principles for re-usable ontologies can be found in [8]. Using mature methodologies for ontology engineering [9] can help to avoid some of the issues which we encountered during the creation of the domain ontologies.

We understand an ontology as loose collection of related concepts. The notion of *related* is deliberately underspecified: it depends on the client's context which particular representation of an existing vocabulary is considered suitable. We discuss the idea of *profiling concepts* to support conceptualizations conflicting with common sense. This phenomena appears not only in-between different information communities, but also between experts of the same domain. Profiling allows for individual interpretations without losing consistency with the underlying ontology. The presentation of a conceptually simple approach to realize profiling for concept descriptions is the main contribution of this paper.

Profiling supports context-sensitive ontology modularization, and various related work on this subject exists. Most research is focussing on the formal definition of modules in ontologies [10, 11], with focus on describing how to define (and how to separate) modules. In [12], the authors introduce a formal way to link the different modules. D'Aquin et al. [13] discuss different aspects ontology modularization methods have to consider (and can be evaluated against). According to [14], the following three approaches for ontology modularization exist: (1) Query-based methods, (2) Network partitioning, and (3) Extraction by traversal. Their segmentation approach for large-scale ontologies is based on the traversal in the ontology graph. The same is true for the implementation presented here.

The following section 2 lists the reasons explaining why we implemented our own version of an ontology repository, and why existing solutions did not meet our requirements. This section will also introduce a use case which acts as a running example for the remainder. In section 3 we discuss our approach

and accordingly the implementation. We introduce the concept of profiling and addressing concepts and how to select relevant collections of concepts in the repository. We conclude the paper with a short evaluation and a summary.

2 Creating and sharing vocabularies on the Web

Authoring sophisticated ontologies in collaboration with domain experts, and making the results accessible on the Web, is only a first step. The active use of these ontologies by other parties (ideally from a different information community) is also needed to enable integration across information communities. Several issues have to be considered to not only complete the first, but also the second phase. In the following section we discuss our experience in knowledge acquisition and ontology engineering, and list the problems encountered which eventually resulted in the implementation of CORE.

2.1 Building Domain Ontologies

Deciding if one particular site may be a suitable location for quarrying mineral resources relies on a variety of criteria. The acquisition, analysis, and presentation of potentially relevant information guiding the decision maker has been the subject of the research project SWING[1]. The relevant information is served by Web services which have been semantically annotated with domain ontologies. The whole process (discovery,pre-processing, and rendering) has been implemented as a workflow. Such Web service compositions were also the focus of the GDI-Grid[2] project. Here, ontologies are used for the semantic validation of the Web service workflows. In SWING we mainly focused on interviews with domain experts like geologists to capture the relevant concepts and their properties [15]. The result of the conceptual phase [9] were extensive concept maps representing the core concepts which had to be implemented in the ontologies. Figure 1 depicts a small excerpt of one concept map. Graphical tools for authoring concept maps ship with two interesting features: Colour has been used to organize the concepts which belong to the same domain. The concept's spatial distribution is particular useful to group those which are in some way related (without the need to explicitly associate them with a domain using colour). Unfortunately, colour and location can not be directly re-used for the implementation of the concept maps as ontologies.

Figure 1 represents the engineer's view of the concept *River*. For computing the river's discharge (the product of the stream velocity and cross-sectional area), environmental models for flood prediction make use of information about the underlying terrain and observations coming from sensors. Computing the cross-sectional area relies on detailed information about the *Depth* of the river. In the remainder of this paper, we are using this particular quality as a running

[1] Project results and videos are available at http://www.swing-project.org/
[2] On-going project, more information available at http://www.gdi-grid.de/

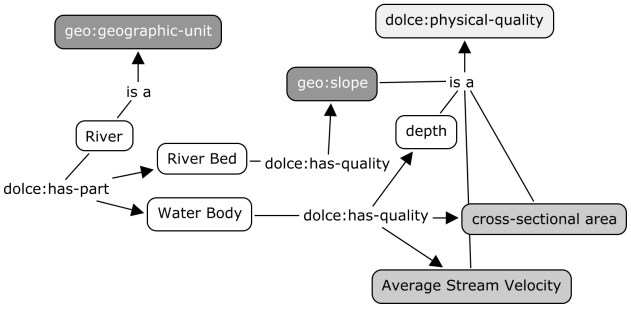


Fig. 1. An excerpt of a concept map

example to explain the idea of domain-independent conceptualizations. The captain steering a vessel through the river has one particular view on the river's depth. He is only concerned about the minimum depth of the official water way. The biologist may be more interested in maximum freezing depth, which allows for modelling the winter conditions for the fish population. In the next section we discuss the problems encountered when we realized that we have to integrate such different perspectives into the domain ontologies.

2.2 Conflicting Conceptualizations

The results of the knowledge acquisition where captured using either tools for building concept maps or by writing protocols of the discussions with the domain experts. For implementation, these intermediate, sometimes inconsistent, and rather informal models had to be transformed into formal ontologies. The problems described in this section have been the motivation for the implementation of the concept repository.

Until now, a concept has been merely described is by a name and relations to other concepts. Ontologies are meant to serve as formal specifications which explicitly describe the concept. These concept descriptions comprise a name, properties (including relations to other concepts), and additionally axiomatic statements which further constrain the properties. If an ontology is lacking ambiguities, e.g. due to homonyms, naming is not necessarily an issue. Hence, re-using the concept's name as part of its identifier is a common approach suitable for simple ontology building tasks. Since we refer to RDF-encoded ontologies shared on the Web, an identifier is implemented as Internationalized Resource Identifier (IRI). An IRI comprises a namespace and a local name. Concepts are often defined through other concepts: a concept description for river may be identified using the term "River", the river's depth as "RiverDepth", and one particular conceptualization even as "MinimumRiverDepth". This approach does not scale well for extensive ontologies, and eventually results in arbitrarily chosen local names which do not reflect the actual name of the concept. This becomes even more apparent if multiple names in different languages are to be supported for

one concept. By decoupling the identifier's local name from the concept's name, ontologies can support different descriptions with similar names, as well as descriptions with different names. Within CORE, the concept descriptions have automatically generated local names and one common namespace. We make use of Dublin Core [16] metadata properties - in this case `dc:title` for the concept name - to model the identity of a concept description.

The problem of finding appropriate identifiers becomes more apparent if two concept descriptions within the same namespace (which means, the same ontology) describe the same concept. Figure 2 comprises two examples for a description of the concept *River* using the Manchester Syntax [17] of the Web Ontology Language (OWL).

Example 1 (Captain's Perspective).

```
Class: River
  Annotations:
    dc:title "River"

ObjectProperty: has-depth
  Annotations:
    dc:title "has depth"
  Domain:
    River
  Range:
    minimum-depth

Class: minimum-depth
  Annotations:
    dc:title "minimum depth"
```

Example 2 (Biologist's Perspective).

```
Class: River
  Annotations:
    dc:title "River"

ObjectProperty: has-depth
  Annotations:
    dc:title "has depth"
  Domain:
    River
  Range:
    maximum-freezing-depth

Class: maximum-freezing-depth
  Annotations:
  dc:title "maximum freezing depth"
```

Fig. 2. Two different conceptualisation of river depth.

Depending on the context, either of these two descriptions can be considered to be valid. Both share an identical extension since they refer to the same real world concept. The captain's understanding of *River* may differ from the biologist's, but both use the same term to refer to it. Hence, even though concept descriptions have conflicting properties, their names are identical. Modularization – splitting the ontology up into modules with different namespaces – may provide a solution for conflicting concept descriptions. During the implementation we regularly dealt with concepts whose context were not clearly defined. Such border-line cases can not be clearly associated with one particular domain. The need for modularization forces the ontology engineer to also explicitly assign context to concepts which are either context-free or belong to multiple domains. The *maximum-freezing-depth* of a river may be important in the scope of a Biology domain ontology, but is obviously also related to Hydrology. The concept *Depth* itself is domain-independent. Adding such concepts to one specific domain

ontology strengthens the association to the domain, but also impairs re-usability in other contexts. During implementation we decided to interpret modularization differently: ontologies are no longer collections of concepts manually compiled by the ontology engineer. Ontology membership is simply a property itself. Every concept description may be defined to be part of multiple ontologies, and membership can change dynamically. Similarly to the concept's name, we use of the `dc:subject` property to express a concept's membership in a certain domain.

In SWING, the individual modules represented only a small excerpt of the needed vocabulary, and the aggregated graph was much too extensive for the visualization. Sophisticated query techniques for RDF-based vocabularies exist, but relying on such complex solutions impedes re-usability for generic clients. It would then be the client's responsibility to (a) study the ontology to learn how to formulate the query and (b) execute queries where in fact only one URL should be required for selecting the relevant collection of concepts. It should be possible to construct URLs which not only uniquely identify concept descriptions, but also allow for selecting collections of concept descriptions which are in some sense related and therefore important to the client. descriptions are only valid within a one particular domain, since there exists another conflicting description. In our case, the concept River may be modelled to have a quality "'depth"', which is commonly understood as the average depth measured by a gauge.

2.3 A first implementation

Existing solutions like the Tones Ontology Repository[3], Oyster[4], or Pronto[5] are focused on the ontologies as a subjects of interest, not the individual concept descriptions. A first implementation of CORE was released in late 2008 for the SWING project [18]. Only some of the features discussed in this paper have been realized in this version. In fact, most requirements were identified during its implementation and use.`dc:title` labels the concepts, and the concatenated language tag, e.g. "@en", marks different languages. The namespace defines the scope of the needed ontology. For example, the URL "http://.../core/GDI-Grid/" has been used to request all concepts associated with the "GDI-Grid" domain. The URL "http://.../core/Acoustics/GDI-Grid/" retrieves an intersection of two domains: the result is a list of concepts which have been defined valid for both domains.

The focus on using namespaces for defining scope had one major drawback. Managing the import of namespaces for local ontologies became a tedious task, since nearly every concept description was defined in a different scope. Additionally, the separation between listing all the concepts in one context (only the namespace is used as the URL) and concept description was not accepted by the

[3] See: http://owl.cs.manchester.ac.uk/repository/
[4] See: http://oyster.ontoware.org/
[5] See: http://metadata.net/sfprojects/pronto.htm

users. Hence, a new implementation of the concept repository[6] was initiated, which is currently in active development.

3 Solutions

In the following section, we introduce some suggestions to overcome the encountered problems. These includes the notion of profiling concepts to model the domain-specific perspective without breaking the relation to the original concepts, a solution for selecting subsets of concepts which may be relevant according to certain criteria, and the idea of regular consistency checks for the relationships between concepts.

3.1 Profiling concepts

The example of Figure 2 listed two valid, but conceptually inconsistent, descriptions for the concept *River*. Following Guarino, we consider a concept description (and its associated ontology) to be a "logical theory which gives an explicit, partial account of a conceptualization" [19]. The ontology engineer's subjective view on reality can only result in a partial description. Different perspectives on one concept may result in diverging and sometimes conflicting descriptions. The object's identity criteria are defined through its characteristic properties [20].

Profiling concepts supports different viewpoints on concepts within one ontology. It allows for refining and extending conceptual structures, without losing the applicability of the underlying model [21]. One profile[7] concretises another concept description. The concept itself is always domain-independent; the same is true for characteristic properties. The profile extends (and therefore concretises) a domain-independent description by either refining existing or adding new properties. Profiling is not inheritance. Both, source and profiled concept description, refer to the same concept. Both have an identical extension. All instances of *River* are also instance of *River (Biology)*. But only some instances of *River* can be considered to also be a *Creek* (which is modelled as sub-class of *River*) A profile specifies one particular viewpoint on a concept, but it does not affect its extension. Accordingly, both share the same name for identification.

Figure 3 illustrates how *River* has been refined to reflect the biologist's perspective. It also shows how concept descriptions are stored within the repository. During import, the identifiers are automatically generated (as hexadecimal codes) from the title and, if existing, the subject. A concept is profiled by specifying that a property is only valid within a certain context, i.e., a `dc:subject` annotation is added. An existing property is refined by additionally re-using the source property's `dc:title`-annotation and changing the property's range. In the

[6] This time as part of an open source project. More information is available at http://purl.org/net/sapience/docs/. All source code is publicly accessible via the subversion repository.

[7] The idea of "profiling" is commonly used in the standards communities to explain if one standard is concretising another.

```
Class: 26c623af
  Annotations:
    dc:title "River"
    rdfs:seeAlso River_Biology
  SubClassOf:
    geo:geographic-object

DatatypeProperty: 1a50ca0c
  Annotations:
    dc:title "has depth"
  Domain:
    26c623af
  Range:
    double
```

```
Class: 618c2089
  Annotations:
    dc:title "River"
    dc:subject "Biology"

ObjectProperty: addac6d
  Annotations:
    dc:title "has depth"
    dc:subject "Biology"
  Domain:
    618c2089
  Range:
    d1ce83e7

Class: d1ce83e7
  Annotations:
    dc:title "maximum freezing depth"
    dc:subject "Biology"
```

Fig. 3. The concept *River* (left) and the profiled concept *River (Biology)* (right)

figure, the domain-independent concept description includes the property "has Depth" with a literal as its range. The range of this property has been changed and refers to the "maximum freezing depth" for the profiled concept. The ontology engineer is responsible for creating the profiled concept *River (Biology)*, adapting the properties, and adding a `rdfs:seeAlso` annotation to link the original concept description to the new profiling description. We already mentioned that semantic heterogeneities may not only exist between different information communities, but already within one community, or even within one organization. Profiles can again be source descriptions for other profiles. The transitive nature of profiling enables individual conceptualizations at all levels, with the option to trace the profiles back to the original source. Users are then able to navigate to the profiled concept descriptions if needed. In the following section we explain how to retrieve the concept descriptions either for *River* (without the refined properties) or *River (Biology)* (the value of the `rdfs:seeAlso` annotation in the figure).

3.2 Accessing concepts

Internally, all concept descriptions have automatically generated local names which are used for identification. The concept's identity, on the other hand, is defined by its title and subject. Title and subject can be defined in various ways in the URL. The expressions `River_Biology`, `/subject/Biology/River`, `River?subject=Biology` and `/describe?title=River&subject=Biology` all identify the same concept description. Only one context can be specified in the URL. The first three examples are internally transformed into the fourth. In the end, the query task (see section 3.3) *describe* is triggered with the parameters

title and *subject*. The result of this query is the concept description listed in figure 4. The resulting RDF is formatted according to the requested URL. As suggested in [22], the concept description's identifier is always the request URL. The style of the other resource identifiers in the concept description, e.g. for the properties, is equivalent to the style of the request URL. If a concept description with the given parameters does not exist, the result is either a redirection (HTTP Response Code 303) to the potentially correct concept description (e.g. if a non-existent domain-dependent concept is requested, a redirect to the domain-independent description is returned) or an exception (HTTP Response Code 404).

```
Class: River_Biology
  Annotations:
    dc:title "River"
    dc:subject "Biology"

ObjectProperty: has-depth_Biology
  Annotations:
    dc:title "has depth"
  Domain:
    River_Biology
  Range:
    maximum-freezing-depth
```

Fig. 4. Result for the URL "http://.../rdf/River_Biology"

3.3 Accessing ontologies

An ontology is a collection of related concepts descriptions. Depending on the user's need, the type of the relevant relation may differ. In most cases, though, she might be interested in all concepts associated with one domain, e.g. Biology or Hydrology. Which specific collection, and therefore ontology, is returned by CORE depends on the ontology identifier. As when accessing a concept description, the access to an ontology is defined by the URL parameter. For example, the query action *describe* returns a concept description matching the given query parameters *title* and *subject*. We distinguish between query and update actions. The first triggers a SPARQL [23] query to the internal RDF repository (based on Sesame [24]), and optionally transforms the result. The latter is used to upload ontologies into the repository. We have developed the query tasks *all* and *neighbors* for CORE. The neighbourhood of one particular concept comprises all other concepts which are directly related to the query concept via its properties. If the optional depth-parameter is specified, the properties of the related concepts are considered as well. The all-action returns all concept descriptions which have the given subject-parameter defined as their domain. As for the describe-action,

all actions support the encoding either in the URL's path (RESTful approach), or in the query fragment. Figure 5 shows three equivalent URLs to return all concepts within the domain Biology.

```
(1) http://.../rdf/Biology/
(2) http://.../rdf/subject/Biology/
(3) http://.../rdf/all?subject=Biology
```

Fig. 5. Constructing the Ontology identifier.

The idea of query actions is not constrained to the two introduced actions. The action *is-similar* could return similar (but not explicitly related) concepts, the action *has-property* may query for all concepts with the given property. Even though we've implemented CORE as a repository for ontologies, it might also be deployed for other RDF-based vocabularies. Using it, for example, as a gazetteer would require query actions supporting spatial queries like *contains*, which runs not only a SPARQL query, but also performs spatial filtering.

4 Evaluation

The first version of the concept repository has been evaluated in two research projects. The new features discussed here were implemented and tested (using module tests), and will be released in the next version. It is deployed as software as a service using the Google infrastructure, which addresses issues such as scalability, performance, and sustainability [25]. Scalability for RDF repositories is primarily concerned about performance of handling very large numbers of triples. Since reducing the amount of retrieved ontology elements has been the objective of CORE, scalability regarding the extent of the ontologies was not investigated. Other open issues, i.e. the scalability of the profiling approach, have to be tested in a long-term evaluation, which is planned in the just started research project ENVISION (http://www.envision-project.eu).

5 Conclusion

Language independence has always been one of the key requirements. The ontologies in SWING and GDI-Grid were implemented using the Web Service Modeling Language WSML[8]. Support for the more popular OWL Web Ontology Language has been identified as a requirement as well. CORE is not restricted to one particular ontology language, but requires an RDF-encoding. CORE is a sophisticated solution to access resources in an RDF repository.

[8] More information available at: http://www.wsmo.org/wsml/

In this paper, we presented an implementation of an ontology repository. We discussed why ontologies published on the Web are rarely re-used in semantically enriched applications, and listed the problems we encountered during knowledge acquisition and ontology engineering. Our proposal to facilitate the use of existing shared vocabularies included the following recommendations: profiling of concepts supports adaptation of existing domain concepts to local needs, without losing the alignment to the underlying domain ontology. A RESTful approach to access the shared vocabularies simplified local integration. Domain-specific information can simply be encoded in the URL used to identify a concept. Continuously running consistency checks test the relationships within the ontologies to ensure valid connections. If we are able to facilitate re-usability of existing shared vocabularies, the envisioned semantic integration of data across information communities may become reality. We believe the presented implementation of the concept repository CORE can contribute to this vision.

6 Acknowledgments

The presented research has been funded by the BMBF project *GDI-Grid* (BMBF 01IG07012) and the European projects *SWING* (FP6-026514) and *ENVISION* (FP7-249120).

References

1. Kuhn, W.: Geospatial Semantics: Why, of What, and How? Journal on Data Semantics III **3534** (2005) 1–24
2. Maué, P., Ortmann, J.: Getting across information communities. Earth Science Informatics **2** (2009) 217–233
3. Hayes, P.J., Halpin, H.: In Defense of Ambiguity. International Journal on Semantic Web & Information Systems **4** (2008) 1–18
4. Berrueta, D., Phipps, J., Miles, A., Baker, T., Swick, R.: Best Practice Recipes for Publishing RDF Vocabularies (2008)
5. Handschuh, S., Staab, S.: Annotation for the Semantic Web (Frontiers in Artificial Intelligence and Applications). IOS Press (2003)
6. Graves, M., Constabaris, A., Brickley, D.: FOAF: Connecting People on the Semantic Web. Cataloging & classification quarterly **43** (2007) 191–202
7. Raskin, R., Pan, M.: Knowledge representation in the semantic web for Earth and environmental terminology (SWEET). Computers & Geosciences **31** (2005) 1119–1125
8. Smith, B.: Against Idiosyncrasy in Ontology Development. In: Proceeding of the 2006 conference on Formal Ontology in Information Systems, Amsterdam, The Netherlands, The Netherlands, IOS Press (2006) 15–26
9. Gomez-Perez, A., Corcho, O., Fernandez-Lopez, M.: Ontological Engineering : with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. First Edition (Advanced Information and Knowledge Processing). Springer (2004)
10. Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: Modular reuse of ontologies: theory and practice. J. Artif. Int. Res. **31** (2008) 273–318

11. Grau, B.C., Parsia, B., Sirin, E., Kalyanpur, A.: Modularity and web ontologies. In: In Proc. KR-2006. (2006) 198–209
12. Bao, J., Caragea, D., Honavar, V.: On the semantics of linking and importing in modular ontologies. In: The Semantic Web - ISWC 2006. (2006) 72–86
13. d'Aquin, M., Schlicht, A., Stuckenschmidt, H., Sabou, M.: Ontology modularization for knowledge selection: Experiments and evaluations. In Wagner, R., Revell, N., Pernul, G., eds.: Database and Expert Systems Applications. Volume 4653 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg (2007) 874–883
14. Seidenberg, J., Rector, A.: Web ontology segmentation: analysis, classification and use. In: WWW '06: Proceedings of the 15th international conference on World Wide Web, New York, NY, USA, ACM (2006) 13–22
15. Schade, S., Maué, P., Langlois, J., Klien, E.: Knowledge acquisition with geologists - a field report. In: ESSI1 Semantic Interoperability, Knowledge and Ontologies, EGU General Assembly 2008. (2008)
16. Weibel, S.L., Koch, T.: The Dublin Core Metadata Initiative: Mission, Current Activities, and Future Directions. D-Lib Magazine **6** (2000)
17. Horridge, M., Drummond, N., Goodwin, J., Rector, A.L., Stevens, R., Wang, H.: The Manchester OWL Syntax. In Grau, B.C., Hitzler, P., Shankey, C., Wallace, E., Grau, B.C., Hitzler, P., Shankey, C., Wallace, E., eds.: OWLED. Volume 216 of CEUR Workshop Proceedings., CEUR-WS.org (2006)
18. Schade, S.: D3.3 Ontology Repository with ontologies. Technical report, University of Münster (2008)
19. Guarino, N.: Formal Ontology and Information Systems. In Guarino, N., ed.: Proceedings of FOIS'98, Trento, Italy, 6-8 June 1998., Amsterdam, IOS Press (1998) 3–15
20. Guarino, N., Welty, C.: Identity, Unity, and Individuality: Towards a Formal Toolkit for Ontological Analysis. In: Proceedings of the 14th European Conference on Artificial Intelligence (ECAI), IOS Press (2000) 219–223
21. Koutsomitropoulos, D.A., Paloukis, G.E., Papatheodorou, T.S.: Semantic application profiles: A means to enhance knowledge discovery in domain metadata models. Metadata and Semanticss (2009) 23–33
22. Sauermann, L., Cyganiak, R., Völkel, M.: Cool URIs for the Semantic Web (2007)
23. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. Technical report, W3C (2008)
24. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In: The Semantic Web ISWC 2002. Lecture Notes in Computer Science. Springer (2002) 54–68
25. Erdogmus, H.: Cloud computing: Does nirvana hide behind the nebula? IEEE Software **26** (2009) 4–6

Ontology Recommendation for the Data Publishers*

Antoine Zimmermann[1]

Digital Enterprise Research Institute
National University of Ireland, Galway, Ireland
firstname.lastname@deri.org

Abstract. We present a process for recommending Web ontologies that exemplifies quality, based on criteria that stimulate their reuse and spreading among data publishers: best practices, support by publishers and applications. The quality of ontologies can be assessed in a semi-automatic peer-review process.

1 Introduction

RDF data publishing on the Web has gathered momentum in the last few years, thanks to a general effort to link open data all over the Web. Yet, this trend is slowed down by the difficulty to find appropriate terms for the data to be described. Indeed, apart from a handful of well known ontologies, there are no readily available, easily findable vocabularies for most of the domains that would be good candidates for publishing data in a standard, linkable way.

The typical problems faced by would-be data publishers are: (1) ontologies defining the domain of interest do not exist; (2) they exist but are difficult to find because developed by small groups for experimentation, lacking advertisement; (3) they exist and can be found but they are of poor quality, not complying with standards or best practices; (4) they exist and can be found but there are too many, of mixed quality, and it is difficult to assess which ones are appropriate for a specific use case.

To address the first issue, user-friendly ontology editors have been developed Swoop[1], Protégé[2], etc. But this mostly requires that ontology and domain experts publish more and more data and terminologies. We assume that this will naturally happen when Linked Data and Semantic Web technologies will reach a critical mass which trigger a virtuous circle. We will not address this issue here.

The second item is somehow addressed by Semantic Web search engines. Several of them have been proposed, such as Swoogle [1], Sindice [2], SWSE [3],

* I would like to thank the Pedantic Web Group (http://www.pedantic-web.org/) for their useful discussions, and more particularly Stéphane Corlosquet, Richard Cyganiak, Renaud Delbru, Alexandre Passant and Axel Polleres. This work is partly funded by Science Foundation Ireland (SFI) project Lion-2 (SFI/08/CE/I1380).

[1] http://code.google.com/p/swoop/

[2] http://protege.stanford.edu/

FalconS [4], Watson [5], OntoSearch [6], Ontosearch 2 [7]. Besides, ontologies can be gathered together in repositories [8, 9] that provide additional functionalities for maintaining them. This can address to some extent the third and fourth items because voluntarily submitted ontologies are more likely to be considered by their authors as being of sufficient quality rather than ontologies randomly retrieved from the Web. Moreover, the implemented functionalities may help correcting possible errors and eventually would only display formally valid terminologies. To address the last issue, it has been proposed to add, *e.g.*, Web 2.0-like rating and voting functions to search engines and repositories (*e.g.*, Revyu [10]). However, reviewing ontologies needs advanced knowledge in fields that the ontology users may not have, and the ontology experts are not necessarily inclined to judge ontologies in the same way as social website assess conversations, products, etc.

Therefore, we believe that to guarantee access to the ontologies that are available, well supported and of quality, there is a need for promoting them more actively. In this paper, we argue in favour of having a committee of experts analyse Web vocabularies with respect to their suitability as reusable terminologies for data publishers. As a result of this analysis and evaluation—which can be partly automatised—the committee would advertise the ontology as a "quality vocabulary" and recommend it for describing information in the field applicable to such terminology.

To present this idea in more details, we first discuss the criteria that a Web terminology should fulfil to be labelled as "quality vocabulary" (Section 2). Then, we describe a possible approach to implement such an evaluation and recommendation framework (Section 3). Finally, we show how this integrates with ontology repositories (Section 4).

2 Criteria for a recommended Web vocabulary

Our objective in recommending Web vocabularies is focused on helping data publishers to find adequate terms for describing their data. For this reason our proposed initiative distinguishes itself from other ontology evaluation activities that focus more on engineering, design and logical issues. Moreover, we do not pretend to assess the quality of the modelling of the domain of interest, which could only be judged by a domain expert. Also, we encourage small, lightweight ontologies, which are easier to assess, reuse and scale. In this section we discuss possible criteria for quality vocabularies, recommended for reuse over the Web of Data. More precisely, data publishers would expect vocabularies that are: (1) justified by use cases; (2) easy to reuse and publish; (3) well interoperable with published Linked Data. We analyse these requirements to determine the criteria for quality vocabularies.

Justifying the existence of the vocabulary. As a primary requirement, a vocabulary should be accompanied by a statement about its utility. This includes a general description of the vocabulary and its scope as well as, more importantly, its related use cases. To avoid too much subjectivity in deciding

the relevance of a vocabulary in terms of usage, it can be required that a Web vocabulary proposed for recommendation should be supported by at least some data publishers. We consider this criteria a very important one and would not recommend a vocabulary, be it very well designed, if nobody considers using it. Usage should not be restricted to a unique dataset, not even to a big one by a major player in the field. At least two independent parties should be using the terms, or there should be strong evidence that the terms will be used by several distinct publishers in the near future. As an alternative proof of relevance, the vocabulary publisher could claim potential adoption by showing precise examples of possible usage. *E.g.*, a geo-location vocabulary can be proven to be useful if the authors show that translating existing geographic databases into linked data can be done in an easy and straightforward way to create and publish multiple datasets at a low cost. Finally, the utility of the vocabulary can be demonstrated if existing applications are usefully exploiting the associated data. This can justify the recommendation of a vocabulary since all data conforming to it will interoperate with those existing applications.

Ease of reuse and publication. Since one of the goal of recommending vocabularies is to increase interoperability by reducing the number of heterogeneous terminologies, it is important that the vocabulary be reusable as easily as possible. To achieve this, the content should be made understandable by non-ontology experts. Thus, one of the criteria is the presence of clear labels and textual descriptions for each term in the ontology. Moreover, granularity and complexity should be in line with the use cases. Highly expressive or too specific ontologies should be discouraged if they are not seriously justified. Finally, publication of data conforming to the proposed vocabulary should be made easier, *e.g.*, by providing tools that automatise (partly or totally) the publishing process. For instance, FOAF and SIOC exporters make the creation of online community RDF metadata fully automatic when integrated in a content management system.

Interoperability. To ensure better interoperability, several guidelines have to be followed. Obviously, vocabularies should be published in a standard format, namely RDF(S) and OWL. Additionally, since vocabularies are themselves part of the Linked Data, they should follow the best practices in the field [11]. This includes, *e.g.*, URI dereferencability, entity naming schemes, or authoritativeness of term definition. A term definition is considered "authoritative" if it describes an entity which is in the namespace of the document describing it. Most of these practices can be checked automatically, using, *e.g.*, RDF:Alerts[3]. Moreover, vocabularies are also used to reason about the data, so special care must be taken with this respect. It is desirable to enable interoperability of the vocabulary with both OWL tools and RDFS tools. On the one hand, an OWL ontology can be made more RDFS-friendly by defining all classes as both `owl:Class` and `rdfs:Class`. Similarly, properties defined as `owl:ObjectProperty`, `owl:DatatypeProperty` and `owl:AnnotationProperty` should also be defined as

[3] `http://swse.deri.org/RDFAlerts/`

`rdf:Property`. On the other hand, RDFS terminologies should declare each term as either of the aforementioned types, unless a strong justification comes from the use cases. For instance, the Dublin Core vocabulary does not specify the type of its properties in order to preserve flexibility. Also, an OWL ontology should be kept compatible with OWL DL as much as possible, and any exception should be justified. More generally, vocabularies should use the least expressive fragment of OWL that fulfils the desired requirements.

3 Implementing the recommendation process

This section shows how a framework for recommending Web vocabularies could be implemented in practice.

We notice that most ontologies and Web vocabularies, especially the most popular ones, are developed by academic researchers (FOAF, SIOC, Good Relations, Music Ontology, etc.) Thus, it would be possible to incite the ontology builders to publish their creation through our quality assessment process by establishing a regular submit/review/accept-reject process. An ontology or terminology for data publishing offers a solution to a problem or fulfil a certain need. It can thus be seen as a scientific contribution. We propose to have a call for vocabularies with a review process and editorial constraints.

First, an automatic tool will verify the compliance of the submitted vocabularies to the well established criteria mentioned in Section 2. If these are matched, then a peer-reviewing process will be undertaken by the committee, considering what has been discussed in the previous section. A submitted ontology must be accompanied by a description that will be published together with the ontology. The description—which can take the form of an article—must explains the purpose of the ontology—not only its domain but also its scope and granularity as well as possible or existing applications using it. It must show the utility and the need for such a vocabulary. This is partly proven by the fact that existing (independent) datasets are already using the terms or publishers have committed to use it in the near future.

The descriptions should be kept understandable by non-ontology specialists, and technical details can be given if, and only if, it contributes to showing the utility and interoperability of the vocabulary. As a result of this process, the ontology is either deemed not suitable for recommendation or recommended as a "quality vocabulary". Rejected ontologies can be improved and resubmitted later, eventually leading to better quality of the vocabularies. A centralised Web site would advertise these ontologies, provide documentation about them and allow searching and browsing them. Such a central place could be an existing ontology repository, as discussed in the next section.

4 Integrating recommendations in an ontology repository

The recommended ontologies should be easily searchable, browsable as well as matchable. These are common tasks made possible by ontology repositories.

Therefore, the recommendation process that we described previously may be integrated into a repository that could possibly include other non-recommended ontologies. However, the recommended ones should be emphasised and all operation should be applicable to the quality ontologies only. Moreover, non-recommended vocabularies could be marked by specific labels indicating that, although they are not recommended, they validate some of the criteria for quality.

5 Conclusion

In this paper, we argued that data publishers should be guided in their choice of vocabularies by actively recommending them ontologies that are considered of quality. The recommendation should be based on criteria that span from support by existing publishers and applications to the compliance to the best practices in this field. Evaluating those criteria could take the same form as an academic publication process, with a review phase. As a first step, we would like to launch a workshop on this topic which would create, we hope, an incentive for researchers to design and publish quality ontologies for the Web of Data.

References

1. Finin, T., Ding, Z., Pan, R., Joshi, A., Kolari, P., Java, A., Peng, Y.: Swoogle: Searching for Knowledge on the Semantic Web. In: Proc. of AAAI 2005, AAAI Press / The MIT Press (July 2005) 1682–1683
2. Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., Tummarello, G.: Sindice.com: a document-oriented lookup index for open linked data. International Journal of Metadata, Semantics and Ontologies **3**(1) (2008) 37–52
3. Harth, A., Hogan, A., Umbrich, J., Decker, S.: SWSE: Objects before documents! In: Proc. of the Billion Triple Semantic Web Challenge. (2008)
4. Cheng, G., Ge, W., Qu, Y.: FalconS: Searching and Browsing Entities on the Semantic Web. In: Proc. of WWW 2008, ACM Press (April 2008) 1101–1102
5. d'Aquin, M., Sabou, M., Dzbor, M., Baldassare, C., Gridinoc, L., Angeletou, S., Motta, E.: WATSON: A Gateway for the Semantic Web. In: Poster session of the European Semantic Web Conference, ESWC. (2007)
6. Zhang, Y., Vasconcelos, W., Sleeman, D.: OntoSearch: An Ontology Search Engine. In: Proc. of AI-2004. BCS Conference Series, Springer (2004)
7. Thomas, E., Pan, J.Z., Sleeman, D.: ONTOSEARCH2: Searching Ontologies Semantically. In: Proc. of OWLED 2007. Volume 258 of CEUR Workshop Proceedings., Sun SITE Central Europe (CEUR) (June 2007)
8. Pan, J., Cranefield, S., Carter, D.: A lightweight ontology repository. In: Proc. of AAMAS 2003, ACM Press (July 2003) 632–638
9. d'Aquin, M., Lewen, H.: Cupboard - A Place to Expose Your Ontologies to Applications and the Community. In: Proc. of ESWC 2009. Volume 5554., Springer (June 2009) 913–918
10. Heath, T., Motta, E.: Revyu: Linking reviews and ratings into the Web of Data. Journal of Web Semantics **6**(4) (2008) 266–273
11. Bizer, C., Cyganiak, R., Heath, T.: How to Publish Linked Data on the Web. web published (July 2007)

Semantic Interoperability Framework for Estonian Public Sector's E-Services Integration

Kalle Tomingas[1], Martin Luts[2,3]

[1]Department of Computer Science, Tallinn University of Technology, Estonia,

[2]Department of Informatics, Tallinn University of Technology, Estonia

[3]ELIKO Competence Centre in Electronics-, Info- and Communication Technologies, Tallinn, Estonia

April 26, 2010

Abstract

E-services, based on automated data exchange in distributed technological and organizational environment, are an effective way to build cross-border, controlled information services. Processes of creation, integration, management, reuse, discovery and composition of e-services are not very efficient without understanding the meaning of information resources. Creation and management of human and machine readable semantics of heterogeneous and distributed information resources are more complicated than coordinated documentation process, and require new interoperability principles, architecture and infrastructure. This paper outlines the idea and architecture of the Estonian semantic interoperability initiative in the public sector. The paper presents a collaborative ontology engineering toolset and repository as a part of interoperability infrastructure, built with Semantic Mediawiki, to manage the semantics of information resources.

Keywords: semantic interoperability, ontology, web service, web service annotation, semantic mediawiki, rdf/rdfs, owl, wsdl/sa-wsdl.

1 Introduction

This paper describes the Estonian public sector's semantic interoperability initiative and outlines the architecture of semantic integration of State Information Systems and e-services. The paper provides a framework, a toolset and a collaborative semantic assets management solution in the Semantic Mediawiki[1] (SMW) environment. We concentrate on the practical implementation of infoware metadata management like web services descriptions, ontologies, classifiers, their creation, import and export, versioning, collaboration and change management. The paper shows how annotation and linking of different assets allows us to realize the semantic interoperability idea, manage the heterogeneous information resources and give them human and machine readable semantics. The given semantics is used for search, mange and re-use of existing information assets components, quality insurance, new web services discovery and composition.

Examples of semantic interoperability initiatives can be found in different EU countries: UK Public Sector Linked Data Government [2], Germany's Deutschland Online[3],

[1] http://semantic-mediawiki.org

[2] http://data.gov.uk/

[3] http://www.deutschland-online.de/DOL_en_Internet/broker.jsp

Italian Public Administration[4], Finnish FinnONTO[5] [2], Estonian Semantic Interoperability Framework [1] or Latvian's Semantic Latvia [3]. Ideas behind those projects vary from human readable semantic descriptions to a large-scale international semantic interoperability infrastructure. Other examples about Pan-European semantic interoperability initiatives are SEMIC (SEMantic Interoperability Centre Europe)[6], led by the European Commission's ISA[7] program and SemanticGov[8], which is targeted to provide integrated public services to citizens at the national or Pan-European level [4]. SEMIC is designed as a brokerage platform for existing third party semantic assets (e.g. classification lists, ontologies, etc.). The aim of SemanticGov is to build the new infrastructure (software, models, services, etc.) necessary for enabling the offering of semantic web services for public administration agencies, within and between the EU countries.

2 Semantic Interoperability Architecture

The semantic interoperability architecture for state information system and registries of Estonia (see Figure 1) consists of the following interrelated components: ontologies and semantically annotated objects supported by policies and guidelines, several processes and workflows, tools, educational activities, PR among others. The Administration System for the State Information System [5]and Semantic Asset Management Environment (SEHKE) are the central tools in the semantic interoperability architecture for the state information system. SEHKE fulfills the following tasks in the semantic interoperability architecture: hosting and publishing of ontologies, infoware's metadata including semantic annotations; serving as a semantic search engine for semantic assets (resources).

SEHKE works to ensure the interoperability of public sector registries and the reuse of technical, organizational and semantic resources. For end-users, SEHKE is the tool for obtaining information about existing services (as well as about service descriptions and the principles of service provision) and apply for the right to use a service or propose the creation of a new service.

One of the main components of the semantic interoperability architecture – as designed for Estonian state registries – is ontology. We use the term 'ontology' meaning "a formal explicit specification of a shared conceptualization for a domain of interest" [6] and "information about how individuals are grouped and fit together in a particular domain". Currently, there are some ontologies in Estonian, which could be used to annotate operations performed by state registries and other objects. The ontology component in our architectural framework is not a monolithic structure – for the purposes of easier, domain-expert driven maintenance, it is divided into subject areas or domains, e.g. "Environment", "Social Affairs". The initial tree for categorizing ontologies is based on the official naming of EU activities. The language used in the semantic description of ontology objects draws from W3C recommendation OWL (Web Ontology Language)[9]. Ontologies are developed and maintained in a distributed manner, but stored and published centrally. The tools to be used for ontology maintenance are not prescribed as long as certain standards are followed (e.g. Collaborative Protégé, Semantic Mediawiki or others). In the next section, we give an overview of a new online collaborative tool for semantic resources management, which is playing an important

[4] http://www.cnipa.gov.it/site/it-IT/
[5] http://www.seco.tkk.fi/projects/finnonto/
[6] http://semic.eu/about_semantic_interoperability_centre_europe.html
[7] http://ec.europa.eu/isa
[8] http://www.semantic-gov.org/
[9] http://www.w3.org/TR/2004/REC-owl-features-20040210

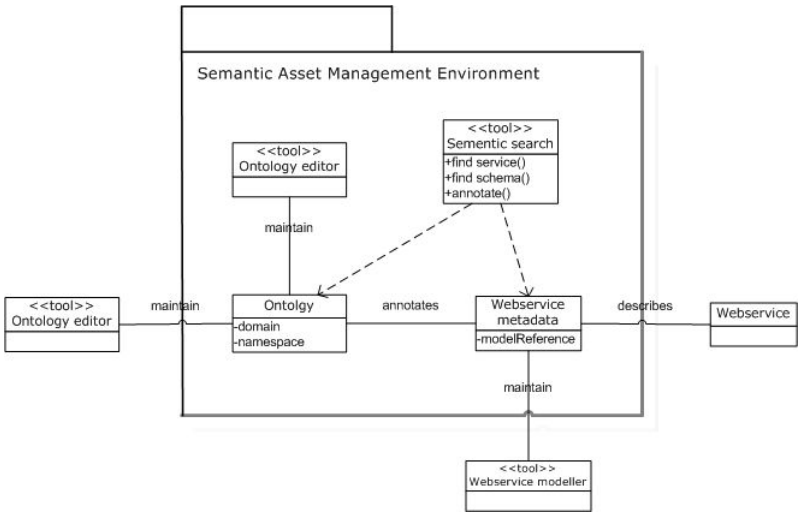


Figure 1: Semantic interoperability conceptual architecture for state information system and registries

part in the large-scale semantic interoperability initiative.

3 Collaborative Semantic Resource Management

SEHKE is a web-based collaborative environment for creating, management and annotation of semantic resources and assets of state information systems and registries.. SEHKE is built to handle information resources like ontologies, concepts and properties, subject areas (domains of interests), web service descriptions and classifications. SEHKE environment is one piece in the big picture of large-scale semantic interoperability architecture. The functionality of the SEHKE toolset covers the creation process of the information assets (e.g. ontologies, web service descriptions, classifiers), the semantic annotation of web service descriptions, collaboration, responsibility, notifications and approval, quality and version control, import and export, semantic search and finally (re)usage of all assets. The SEHKE environment is set up and implemented in the Semantic Mediawiki environment, where the needed functionality is built with semantic extensions, templates and forms, existing plugins and new developed components for special requirements (import and export).

System concepts of the SEHKE environment are based on the main categories of semantic assets. System concepts are implemented as wiki pages, with certain types of mandatory properties and a set of defined forms, to create and manage a content of different concept types. One required property is "Type", which is the main instrument to categorize concepts (e.g. `[[Type::Ontology]]`). Each SEHKE concept has its own set of required or allowed properties that are used to implement the possible relationships between different system concepts (e.g. ontology page has property `[[SubjectArea::Economy]]` to define a relation that the ontology belongs to the subject area). Named and implemented properties and their values form the schema of concepts (Figure 2) that represents the semantic interoperability requirements and the structure of SEHKE functionality.

Ontology is one of the key concepts of the SEHKE tool and it stands for formalized semantics. Ontology is a named set of concepts, their definitions, hierarchies and properties that represents the conceptualization of the domain or subject area, which

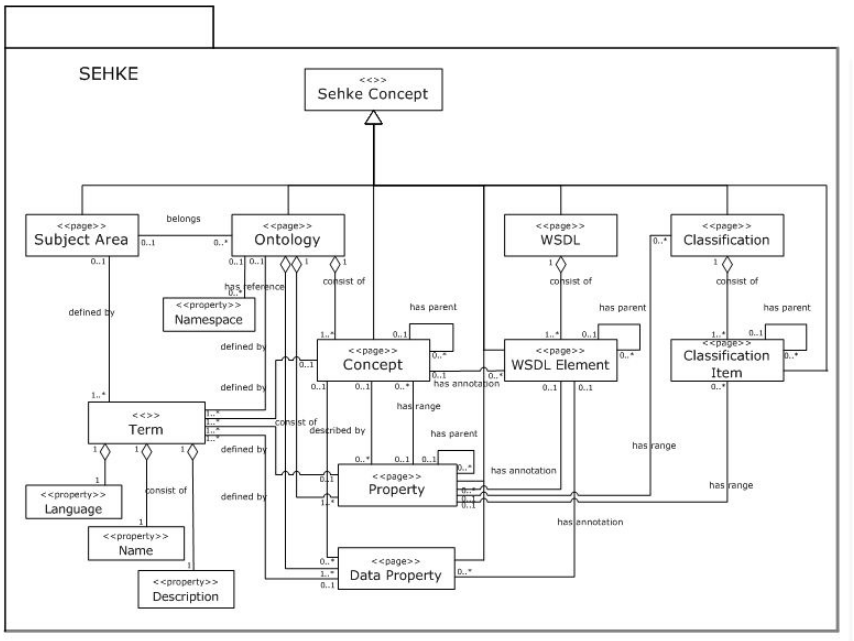


Figure 2: Schema of system concepts of SEHKE environment

is in the interest of the Estonian public sector. Ontology and its concepts are besides web services descriptions and classifications a main type of semantic assets. Ontology belongs to the subject area that frames the domain and the responsibility of the owner of the ontology. Ontology is implemented as a special page, with the idea to group and collect related classes, their properties, instances and rules. Ontology, its member concepts and properties (data and object properties) can have multiple terms (set of names and descriptions) in different languages, which explain the meaning of the item. Concepts (classes) are organized into taxonomies or hierarchies (e.g. using property definition `[[subclassOf::ParentConceptName]]` to associate concept page with a higher level concept), and they can have certain formalization rules that can be automatically checked and validated by automatic reasoning programs. The creation of ontologies, concepts and properties is simplified through automated detection of similar assets, supported by syntactic and semantic search algorithms and backed with the different language techniques. Ontology creation and management has online collaborative support functions, like email notifications, change acceptance and an iterative process for achieving consensus between multiple parties. The collection of all SEHKE pages forms the general wiki namespace (e.g. http://www.sehke.eu/ is the namespace of public instance of the SEHKE environment). In addition to the wiki's namespace, ontology has special "Namespace" property (e.g.), for different namespace values that are used in ontology export/import to rdf/owl. The multiuser collaborative editing, multilingual terms and descriptions and configurable language techniques makes the SEHKE a real multi-national semantic tool.

WSDL is a SEHKE concept for web services descriptions that documents e-services of the Estonian state information system's data transportation layer X-Road (Estonian X-Tee)[10]. X-Road currently supports the technical interoperability of the components of state information systems and registries and SEHKE is like an extension for adding a semantic layer top of the X-Road infrastructure. Web services description documents

[10] http://www.riso.ee/en/information-policy/projects/x-road

are semantic assets, which can be imported and exported in WSDL/SA-WSDL[11] language. The essential idea of semantic interoperability is to add semantic annotations to web services input/output data elements and structures and related operations. Defined ontology's concepts and data or object properties are the key ideas that are used for WSDL annotations. An annotation is a semantic relationship between a WSDL element and an ontology's concept, which can be created manually or semi-automatically (in the future releases), using syntactic and semantic search algorithms, to match similar terms with the concepts (e.g. SAWSDL *modelReference* extension attribute used to associate WSDL elements with a semantic model). Semantic annotation of web services brings us to the next level, which makes available automatic discovery of web services and composition of new web services. The composition of new services is based on existing services, operations and their input/output data elements, which have a semantic match on the conceptual level.

4 Conclusions

The main approach to extend semantic interoperability in communication and information exchange process is to build a higher semantic integration layer on top of participating information systems and organizations. Mapping critical communication structures against ontologies and the same semantic concepts using annotation techniques adds human- and machine-readable semantics to the process. Opening the meaning of system components and functions, integrating computer programs, linguistic techniques and human touch to knowledge management process, we fulfill the promises and expectations of semantic interoperability idea and creating better public sectors services. The SEHKE toolset that is powered with semantic functionality of the SMW is the open platform for collaboration and semantic asset management (e.g. ontologies) and meant to be part of federated pan-European repository initiative: SEMIC.EU [7].

References

[1] Haav, H.M., Kalja, A., Küngas, P., Luts, M.: Ensuring Large-Scale Semantic Interoperability: The Estonian Public Sector's Case Study. Databases and Information Systems V, IOS Press 2009, pp.117-129.

[2] Hyvönen, E., Viljanen, K., Tuominen, J., Seppälä, K.: Building a National Semantic Web Ontology and Ontology Service Infrastructure - The FinnONTO Approach. In Proceedings of the European Semantic Web Conference (ESWC 2008), Tenerife, Spain, 2008, (available at http://www.seco.tkk.fi/publications/2008/hyvonen-et-al-building-2008.pdf)

[3] Barzdins, J., Barzdins, G., Balodis, R., Cerans, K., Kalnins, A., Opmanis, M., Podnieks, K.: Towards Semantic Latvia. In Proceedings of Communications of 7th International Baltic Conference on Databases and Information Systems. Vilnius, Lithuania, 2006, pp. 203-218, (available at http://melnais.mii.lu.lv/audris/CReady_Barzdins_TowardsSemanticLatvia.pdf)

[4] Vitvar, T.: Infrastructure for the Semantic Pan-European E-government Services. In Proceedings of the 2006 AAAI Spring Symposium on The Semantic Web meets e-government (SWEG), 2006, (available at http://www.semantic-gov.org/index.php?name=Web_Links&req=visit&lid=4)

[5] Vallner, U.: Nationwide Components of Estonia's State Information System. In Baltic IT&T Review 3(42), 2006.

[6] Gruber, T.R.: A Translation Approach to Portable Ontology Specifications. In Knowledge Acquisition, Vol. 5, 1993, pp.199-220.

[7] Reichling, K., Luts, M., Fahl-Spiewack, R.: A pan-European repository: SEMIC.EU as the point of reference for eGovernment ontologies, ORES, 2010

100

[11] http://www.w3.org/TR/2007/REC-sawsdl-20070828

Ontology Repositories with Only One Large Shared Cooperatively-built and Evaluated Ontology

Philippe A. Martin
ESIROI STIM, University of La Réunion, France
and adjunct researcher of the School of I.C.T. at Griffith Uni., Australia

Abstract. This article first lists reasons why an ontology repository - or, more generally, a knowledge base (KB) server - should permit the collaborative building of one well organized KB rather than solely be a repository for heterogeneous KBs. To that end, the article proposes a KB editing protocol that keeps the KB free of automatically/manually detected inconsistencies - and leads knowledge providers to semantically organize their terms and statements - while not forcing them to discuss or agree on terminology and beliefs nor requiring a selection committee. Then, the article gives ideas on how to extend this support to allow a precision-oriented collaborative evaluation of each information provider and piece of information.

Keywords: knowledge sharing/integration/retrieval/evaluation

1 Introduction

Ontology repositories are often only imagined as being collections of static formal files (e.g., RDF documents) more or less independently developed, hence loosely inter-connected and mutually partially redundant or inconsistent. Section 2 shows that this "static file based approach" - as opposed to a "collaboratively-built well-organized large knowledge base (cbwoKB) server approach" - makes knowledge sharing/re-use tasks complex to support and do correctly or efficiently, especially in a collaborative way. Most Semantic Web related research works are intended to support such tasks (ontology creation, retrieval, comparison and merging) and hence are useful. However, most often, they lead people to create new formal files - thus contributing to the problems of knowledge re-use - instead of inserting their knowledge into a cbwoKB server. Indeed, it seems that WebKB-2 [10] (webkb.org) is the only ontology server that has protocols supporting governance-free loss-less well-organized knowledge sharing. (There are no such protocols in CYC, Ontolingua, OntoWeb, Ontosaurus, Freebase, semantic wikis ...). WebKB-2 also has a large general ontology and hence has at least two of the elements necessary to build a cbwoKB (Other “shared ontology” servers/editors i) let any authorized users make any change in the shared ontology (this discourages information entering or leads to edit wars), or ii) rely on each user or some privileged users to accept or reject changes made in the shared ontology (this is bothersome for the evaluators, sometimes forces them to make arbitrary selections,

and is a bottleneck in information sharing that can cause long delays or discourage information providers). Section 3 gives protocols - with many yet unpublished ideas - to avoid these governance problems and thus support scalable collaborative building of a cbwoKB, i.e., a KB where detected *partial* redundancies or inconsistencies are prevented or made explicit via relations of specialization, identity and/or correction; thus, in a cbwoKB, each object has one "right place" in the specialization hierarchy and is then easily retrievable and comparable to the other objects. Section 4 gives ideas on how this support can be extended to allow collaborative knowledge evaluation.

2 Approaches Based on Files Versus cwoKB Servers

With files, information retrieval (IR) often leads to a list of possibly relevant files or *pieces of information* (***objects***, e.g., a formal term or a informal sentence) whereas it leads to an exact answer in a cbwoKB or within the content of one formal file. Such an answer may be a portion of the cbwoKB, e.g., a part/subtask/specialization hierarchy (with associated argumentation structures) if the query is of the kind "what are the resources/tools/methods to do ...". Such semantically structured answers allow a user to find and compare all relevant objects instead of getting a long redundant list of objects/files where original/precise ones are hidden among/behind objects that are more general, mainstream or from big organizations. This is also why IR quality decreases when the size and number of the files increases, but not when the number of objects increases in a cbwoKB.

The more objects two files contain, the more difficult it is to link these files via semantic relations and hence to semantically compare, organize and evaluate them. Instead, similarity/distance (statistical) measures have to be used. In a cbwoKB, when needed, semantic queries can be used to filter objects or generate files, according to arbitrary complex combinations of criteria, e.g., about the creators of the objects. (Some of these criteria may be used for the internal organization of the cbwoKB but the resulting "views" or "contexts" are language/representation dependent choices and, unlike (semi-)independently created static files, lead the users to strongly relate objects of different views). Ontology libraries, from the first ones such as the Ontolingua library to imagined ones such as "The Lattice of Theories" [15], are often organized into "minimal and internally consistent theories" to maximize their re-use. However, this also leads to few relations between objects of different ontologies, as well as implicit redundancies or inconsistencies between them, and hence more difficulties to compare, merge or relate them. On the other hand, as acknowledged by the author of [15], if the objects are organized into a cbwoKB, such (lattices of) theories can be generated via queries.

With files, change management requires version management (which leads to more files and many information management complications); not within a cbwoKB, as Section 3 shows.

With formal files as inputs and outputs, knowledge re-use or integration leads to the creation of even more files and requires people to select, compare, relate, merge, adapt and combine (parts of) files. Except for simple applications where fully automatic tools can deliver good-enough results, these are complex tasks that have to be done by trained people who know the domain. Most works in collaborative knowledge sharing or "ontology evolution in collaborative environments" are about (semi-)automatic procedures for integrating two ontologies [5] and for rejecting or integrating changes made in other ontologies, e.g., [2][12][13]. In a cbwoKB, no adaptation or integration has to be done for each re-use: the most important/defining relations from an object to other ones have to be entered by its creators and then they can be incrementally complemented or corrected by any user. Indeed, it is often the case that only the object authors know what their objects really mean or have some other kinds of information required for relating their objects to other ones.

A cbwoKB maximizes the use of principled multi-inheritance hierarchies (for specialization/mereological/spatial/... relations) where each object has one "right place" in the sense that different users would search or insert this object at the same place. Only a KB server with a large cbwoKB can permit a knowledge provider to simply/directly add one new object "at its right place" and guide her to provide precise and re-usable objects that complement the already stored objects. The protocols of the next two sections work only with a cbwoKB.

3 Collaborative Editing of a KB

The next points describe the principles behind the editing protocols implemented in WebKB-2 to make it a cbwoKB server, and make some comparisons with features of RDF (which only supports a personal-file based approach). WebKB-2 allows the use of several knowledge representation languages (KRLs): RDF/XML (an XML format for knowledge using the RDF model), KIF and other ones which are here collectively called KRLX and that were specially designed to ease knowledge sharing: they are expressive, intuitive and normalizing (i.e., they guide users to represent things in ways that are automatically comparable). One of them is named Formalized English (FE). It will be used for the examples.

1. In WebKB-2, every object is a term or a statement (generally, a relation between two quantified terms or some relations within the same *context, i.e., meta-statement*). A term refers to a concept/relation type or an individual (an instance of a first-order type). A statement is an individual and is either informal, formal or semi-formal (when it uses a formal syntax and some terms/objects that are informal or referring to informal/semi-formal objects). A (semi-)formal term is a unique *identifier* for a (semi-)formal object. An informal term is a *name* for an object. Different objects may have common names, not common identifiers.
   Every (semi-)formal object has an associated source: creator or source file. The (unique) meaning of a (semi-)formal object may be left implicit and hence might be known only by its creator. Informal objects may also have an associated creator:

their meanings are those that their source has implicitly given them. These distinctions permit the differentiation of (in-)formal objects and create one specialization/generalization hierarchy categorizing all objects. More precisely, this is an "extended specialization/generalization" hierarchy since in WebKB-2 the classic "generalization" relation between formal objects (logical implication) has been extended to apply to informal objects too.

In KRLX, informal objects are double quoted, and object identifiers are either URIs or include their source identifiers as prefixes or suffixes. This is a common solution to avoid lexical conflicts. KRLX allows the use of shortcuts for a source may be used, e.g., `wn#bird` refers to one of the WordNet categories for the English word `"bird"`. The informal statement `"birds fly"_[u1]` was created by the user u1. A difference with XML name-space prefixes in RDF/XML is that the lexical declaration of a shortcut is also a semantic declaration of a term for the source, thus encouraging the creator of the declaration to specify what the source is (a person, a file, etc.); this is also possible in RDF but is not mandatory.

More importantly, RDF has no notion of "belief" whereas in WebKB-2 each object is, in a sense, contextualized by its source. For example, if a statement S created by a user U is not a definition, it is a belief of U. Similarly, a statement by a user U on another user's statement S2 is actually U's belief on his interpretation of S.

In KRLX, a user also has an easy way to i) represent his belief that certain statements belong to a certain source, or ii) associate a private key with its user identifier to prevent another user to impersonate him, and iii) use an encrypted form of this key (i.e., the related public key) for identifying himself.

A KRL that is meant to support knowledge sharing should offer normalized ways to allow this so that knowledge sharing tools can support reasoning or collaboration based on the knowledge sources. RDF and RDF/XML do not yet offer a standard way to allow this. This will come: SPARQL and N3 already offer a way to specify that a statement belongs to a source.

2. *Any user can add any object* and use it in any statement (as in RDF) but *an object may only be modified or removed by its creator*. This last part has no equivalent in RDF since it is a knowledge model, not a collaboration model.
3. *Each statement has an associated source S, and hence,* ***if it is not a definition of a term created by S, is considered as a belief of S***. When the creator of an object is not explicitly specified, WebKB-2 exploits its "default creator" related rules and variables to find this creator during the parsing. Similarly, unless already explicitly specified by the creator, WebKB-2 uses the "parsing date" for the creation date of a new object. Unless already specified, the creator of a belief is encouraged to add restrictive contextualizing relations on it (at least temporal and spatial relations must be specified).

   A definition of a term T by the creator C of T may be said to be "neither true nor false" or "always true by definition": a definition may be changed by its creator but then the meaning of the defined term is changed rather than corrected. No one (including C) is allowed to state something about T that is inconsistent with the definition(s) of T. A user u1, is perfectly entitled to define `u1#cat` as a subtype

of `wn#chair`; there is no inconsistency as long as the ways `u1#cat` is further defined or used respect the constraints associated with `wn#chair`. A definition associated with T by a source S that is not C is actually a belief of S about the meaning of T. At parsing time, WebKB-2 rejects such a belief if it is found (logically) inconsistent with a definition of T by S.
Universally quantified statements are *not* definitions. Unlike KIF and N3, RDF and OWL do not have a universal quantifier and hence force users not to make the distinction. In WebKB-2, this distinction leads to very different conflict resolution strategies (conflict between two statements of different sources).

- A conflict that involves two definitions by two sources S1 and S2 is a misinterpretation by one of the sources, say S2, of the meaning of a term S1#T created by the other source, and hence is solved by *automatic term cloning* of S1#T, i.e., by creating S2#T with the same definitions except for one and then replacing S1#T by S2#T in the statements of S2. The difficulty is to automatically guess a relevant candidate for S1#T and a relevant definition to remove for the overall change to be minimal. Annex 2 of [11] provides some algorithms to do so in common cases.
- Otherwise, a *loss-less correction* is used (details in Point 6).

4. *If adding, modifying or removing a **statement** introduces an **implicit redundancy** (detected by the system) in the shared KB, **or** if this introduces an **inconsistency between statements believed by the user** having done this action*, this ***action is rejected***. Thus, in the case of an addition, the user must refine his statement before trying to add it again or he must first modify at least one of his already entered statements. An "implicit" redundancy is a redundancy between two statements without a relation between them making the redundancy explicit, typically an equivalence relation in the case of total redundancy and an extended specialization relation (e.g., an "example" relation) in the case of partial redundancy.
In WebKB-2, a statement is seen as a graph with an interpretation in first-order logic and graph matching is used for detecting if one graph (Y) is an extended specialization of the other (X), i.e., if X structurally matches a part of Y and if each of the terms in this part is identical or an extended specialization of its counterpart term in X. For example, WebKB-2 can detect that the FE sentence `` `Tweety can be agent of a flight with duration at least 2.5 hours'_[u2] `` (which means "u2 believes that Tweety can fly for at least 2.5 hours") is an extended specialization (and an "extended instantiation") of both `` `every bird can be agent of a flight'_[u1] `` and `` `2 bird can be agent of a flight'_[u1] ``. Furthermore, these last two statements are respectively extended specializations of `` `75% of bird can be agent of a flight'_[u2] `` and `` `at least 1 bird can be agent of a flight'_[u2] ``. (Similarly, this last graph can be found to be exclusive with `` `no bird can be agent of a flight'_[u3] ``).
Except for the fact that it takes into account numerical quantifiers and measures instead of just the existential and universal quantifiers, the graph matching for detecting an extended specialization is similar to the classic graph matching for a

specialization (or conversely, a generalization which is a logical deduction) between positive conjunctive existential formulas (with or without an associated positive context, i.e., a meta-statement that does not restrict its truth domain). This last operation is sound and complete with respect to first-order logic and can be computed with polynomial complexity if Y has no cycle [3]. Outside this restricted case, graph matching for detecting an extended specialization is not always sound and complete. However, this graph matching operation works with language of any complexity (it is not restricted to OWL or FOL) and the results of searches for extended specializations of a query graph are always "relevant".

The current reasoner used in WebKB-2 detects extended specializations as well as the violation of relation signatures or exclusion relations. Since this reasoner currently does not also use a rule based system or a theorem prover, it is not complete with respect to first-order logic if rules are represented without using specialization relations. However, this is irrelevant with respect to this article since the presented protocols are not related to a particular inference method, they are only triggered (and hence enforced) whenever an inconsistency or a redundancy is detected or not when a new statement is entered.

However, it is important to note that i) the detection of implicit extended specializations between two objects reveals an inconsistency or a total/partial redundancy, and then ii) it is often not necessary to distinguish between these two cases to reject the newly entered object. Extended instantiations are exceptions: since adding an instantiation is giving an example for a more general statement, it does not reveal a redundancy or inconsistency (here, an inconsistent belief or incorrect interpretation of a term) that needs to be made explicit.

It is important to reject an action introducing a redundancy instead of silently ignoring it because this often permits the author of the action to detect a mistake, a bad interpretation or a lack of precision (on his part or not). At the very least, this reminds the users that they should check what has already been represented on a subject before adding something on this subject.

Adding, modifying or removing a term is done by adding, modifying or removing at least one statement (generally, one relation) that uses this term. *A new term can only be added by specializing another term* (e.g., via a definition), except for process types which for convenience purposes can also be added via subprocess/superprocess relations. *A new statement is automatically added by WebKB-2 into the extended specialization hierarchy* via graph matching or, for informal statements, solely based on the extended specialization between the words they include). An automatic categorization may be "corrected in a loss-less way" by any user. A new informal statement must also be connected via an argumentation relation to an already stored statement. In summary, all objects are manually or automatically inserted in the extended specialization hierarchy and/or the subprocess hierarchy, and hence are easy to search and compare.

5. *If adding, modifying or removing (a statement defining) a **term T** introduces an **inconsistency involving statements created or believed by other users*** (i.e., users different from the one having performed this action), ***T is automatically cloned*** to

ensure that its interpretation by these other users is still represented. In the case of term removal, term cloning simply means changing the creator's identifier in this term to the identifier of one of the other users (if this generated term already exists, some suffix can be added). In a cbwoKB server, since statements point to the terms they use, changing an identifier does not require changing the statements. In a global virtual cbwoKB, identifier changes in one server need to be replicated to other servers using this identifier.

In a cbwoKB, it is not true that beliefs and formal terms (or their definitions, as well as what they refer to, e.g., concepts) "have to be updated sooner or later". Indeed, in a cbwoKB, every belief must be contextualized in space and time, as in `` `75% of bird can be agent of a flight' in place France and in period 2005 to 2006'_[u3]``, even though such contexts are not shown in the other examples of this article. If needed, u3 can associate the term `u3#75%-of-birds-fly--in-France-from-2005-to-2006` with this last belief. Due to the possibility of contextualizing beliefs it is rarely necessary to create formal terms such as `u2#Sydney_in_2010`. Most common formal terms, e.g., `u3#bird` and `wordnet1.7#bird` never need to be modified by their creators. They are specializations of more general formal terms, e.g., `wn#bird` (the fuzzy concept of bird shared by all versions of the WordNet ontologies). What certainly evolves in time is the popularity of a belief or the popularity of the association between an informal term and a concept. If needed, this changing popularity can be represented by different statements contextualized in time and space.

6. *If adding, modifying or removing a **belief** introduces an **implicit inconsistency involving beliefs created by other creators***, it is ***rejected***. However, a user may "loss-less correct" a belief (that he does not believe in) by connecting it to a belief (that he believes in) via a ***corrective relation***. E.g., here are FE statements by u2 that correct a statement made earlier by u1:

   `` `every bird is agent of a flight'_[u1] has for corrective-restriction `most healthy flying-bird are able to be agent of a flight' '_[u2] `` and

   `` `every bird can be agent of a flight'_[u1] has for corrective-generalization `75% of bird can be agent of a flight' '_[u2]. ``

   If instead of the *belief* `every bird can be agent of a flight', u1 entered the *definition* `any bird can be agent of a flight', i.e., if he gave a *definition* to the type named "bird", there are two cases:

   - u1 originally created this type (`u1#bird`); then, u2's attempt to correct the definition is rejected, or
   - u1 added a definition to another source's type - say `wn#bird` since this type from WordNet has no associated constraint preventing the adding of such a definition - and hence i) the types `u1#bird` and `u2#bird` are automatically created as clones (and subtypes of) `wn#bird`, ii) the definition of u1 is automatically changed into `` `any u1#bird is agent ``

`of a flight'_[u1]`, and iii) the belief of u2 is automatically changed into `` `75% of u2#bird can be agent of a flight'_[u2] ``.

In WebKB-2, users are encouraged to provide argumentation relations on corrective relations, i.e., a meta-statement using argument/objection relations on the statement using the corrective relation. However, to normalize the shared KB, they are encouraged not to use an objection relation but a "corrective relation with argument relations on them". Thus, not only are the objections stated but a correction is given and may be agreed with by several persons, including the author of the corrected statement (who may then remove it). Even more importantly, unlike objection relations, most corrective relations are transitive relations and hence their use permits better organization of argumentation structures, thus avoiding redundancies and easing information retrieval.

The use of corrective relations makes explicit the disagreement of one user with (his interpretation of) the belief of another user. Technically, this also removes the inconsistency: an assertion A may be inconsistent with an assertion B but a belief that "A is a correction of B" is technically consistent with a belief in B. Thus, the shared KB may (and should) remain consistent.

For problem-solving purposes, i.e., for an application, choices between contradictory beliefs must be made. To make them, an application designer can exploit i) the statements describing or evaluating the creators of the beliefs, ii) the corrective/argumentation and specialization relations between the beliefs, and more generally, iii) their evaluations via meta-statements (see the next point). For example, an application designer may choose to select only the most specialized or restricted beliefs of knowledge providers having worked for more than 10 years in a certain domain. Thus, this approach is unrelated to defeasible logics and avoids the problems associated with classic "version management" (furthermore, as above explained, in a cbwoKB, neither formal terms nor statements have to evolve in time).

This approach assumes that all beliefs can be argued against and hence be "corrected". This is true only in a certain sense. Indeed, among beliefs, one can distinguish "observations", "interpretations" ("deductions" or "assumptions"; in this approach, axioms are considered to be definitions) and "preferences"; although all these kinds of beliefs can be false (their authors can lie, make a mistake or assume a wrong fact), most people would be reluctant to argue against self-referencing beliefs such as `"u2 likes flowers"_[u2]` and `"u2 is writing this sentence"_[u2]`. Instead of trying to formalize this into exceptions, the editing protocols of WebKB-2 rely on the reluctance of people to argue against such beliefs that should not be argued against.

7. Like all descriptions of techniques, statement/creator evaluation techniques are considered as term definitions and are automatically organized into the extended specialization hierarchy. To support more knowledge filtering or decision making possibilities and lead the users to be careful and precise in their contributions, a cbwoKB server must propose "default measures" deriving a global evaluation of each statement/creator from i) users' individual evaluations of these objects, and ii) global evaluations of these users. Details are given in the next section. These

measures should not be hard-coded but explicitly represented (and hence be executable by the cbwoKB) to let each user specialize them for its goals and preferences. Indeed, only the user can find the criteria (e.g., originality, popularity, acceptance, ..., number of arguments without objections on them) and weighting schemes that suit him. Then, since the results of these evaluations are also statements, they can be exploited by queries on the objects and/or their creators. Furthermore, before browsing or querying the cbwoKB, a user should be given the opportunity to set "filters for certain objects not to be displayed (or be displayed only in small fonts)". These filters may set conditions on statements about these objects or on the creators of these objects. They are automatically executed queries over the results of queries. In WebKB-2, like conceptual querying, filtering is based on a search for extended specializations. Filters are useful when the user is overwhelmed by the amount of information in an insufficiently organized part of the KB.

8. The approach described by the previous points is incremental and *works on semi-formal KBs*. Indeed, the users can set corrective or specialization relations between objects even when WebKB-2 cannot detect an inconsistency or redundancy. As noted above, a new informal statement must be connected via an argumentation relation (e.g., a corrective relation) to an already stored statement. For this relation to be correct, this new statement should generally not be composed of several sub-statements. However, allowing the storing of (small) paragraphs within a statement eases the incremental transformation of informal knowledge into (semi-)formal knowledge and allows doing so only when needed. This is necessary for the general acceptance of the approach.

With these editing protocols, each object is connected to at least another object via relations of specialization/generalization, identity and/or argumentation. They permit a loss-less information integration, since no knowledge selection has to be made. They can be seen as enabling a precise asynchronous dialogue between knowledge providers. To sum up, they permit, enforce or encourage people to interconnect their knowledge into a shared KB, while keeping the KB consistent but without having to discuss and agree on terminology or beliefs.

Since the techniques described in this article work on semi-formal KBs and are not particularly difficult for information technology amateurs - since the minimum these techniques require is for the users to set the above mentioned relations from/to each term or statement - they can be used in (semantic) wikis to avoid their governance problems cited in the introduction and other problems caused by their lack of structure. More generally, the presented approach removes or reduces the file-based approach problems listed in the previous section, without creating new problems. Its use would allow merging of (the information discussed or provided by the members of) many communities with similar interests, e.g., the numerous different communities working on the Semantic Web. From an application viewpoint, the approach seems interesting to allow the collaboratively building of states of the art in scientific domains, corporate memories, catalogues, e-learning, e-government, e-science, research, etc.

The hypotheses of this approach are that i) conflicts can always be solved by adding more precision (e.g., by making their sources explicit: different "observations", "interpretations" or "preferences"), ii) solving conflicts in a loss-less way most often increases or maintains the precision and organization of the KB, and iii) different, internally consistent, ontologies do not have to be structurally modified to be integrated (strongly inter-related) into a unique consistent semantic network. None of the various kinds of integrations or mappings of ontologies that I made invalidated these hypotheses.

4 Evaluating Objects and Sources

Many information repositories support free-text/numerical evaluations on objects or files by people and then display them or statistical measures on them. For example, Knowledge Zone [8] allows each of its users to i) rate ontologies with numerical or free text values for criteria such as "usage", "coverage", "correctness" and "mappings to other ontologies", ii) rate other users' ratings, and iii) use all these ratings to retrieve and rank ontologies. Such evaluations have several problems: i) the evaluations are not organized into a semantic network, ii) the above examples of criteria and their numerical values are not about objects in the ontologies and hence do not help in choosing between objects, iii) multi-criteria decision making is difficult since two sets of (values for) criteria are rarely comparable (indeed, one set rarely includes all the criteria of the other set and, at the same time, has higher values for all these criteria), and iv) similarity measures on criteria only permit retrieval of *possibly "related"* ontologies: the work of understanding, comparing or merging their statements still has to be (re-)done by *each* user.

In a cbwoKB, these problems are strongly reduced, since evaluations are on objects and are themselves objects: they are managed/manageable like other objects and are integrated into a network of specialization, correction and argumentation relations. As previously noted, a cbwoKB should provide "default global measures" for the evaluation of each statement/creator (based on each user's individual evaluations) and allow the users to refine it. Here are comments (general ones due to space restrictions) on the global measures that are currently being implemented in WebKB-2.

- A global measure of ***how consensual a belief is*** should take into account i) the number of times it has been re-used or marked as co-believed, and ii) its argumentation structure (i.e., how its arguments/objections are themselves (counter-)argued). A simple version of such a measure was implemented in the hypertext system SYNVIEW [9]. The KB server Co4 [4] had protocols based on peer-reviewing for finding consensual knowledge; the result was a hierarchy of KBs, the uppermost ones containing the most consensual knowledge while the lowermost ones were the private KBs of contributing users. Establishing "how consensual a belief is" is more flexible in a cbwoKB: i) each user can design his own global measure for what it means to be consensual, and ii) KBs of consensual knowledge need not be generated.

- A global measure of ***how interesting a statement is*** should be based on its type (if it has one, e.g., observation, deduction, assumption, preference, ...), on its relations (especially those arguing for/against it or representing its originality, acceptance, ...), and on the *usefulness* of the authors of these relations (see below).
- A global measure of ***the usefulness of a statement*** should exploit (at least) the above two measures.
- A global measure of ***the usefulness of a user U*** should incorporate the global measures of usefulness of U's statements and, to encourage participation in evaluations, the number of objects he evaluated.

Given these comments, the motivation for enabling end-users to adapt the default measures is clear. However it is done, taking into account the above cited elements should encourage information providers to be careful and precise in their contributions and give arguments for them. Indeed, unlike in traditional discussions or anonymous reviews, careless statements here penalize their authors. This may lead users not to make statements outside their domain of expertise or without verifying their facts. (Using a different persona when providing low quality statements does not seem to be a helpful strategy to escape the above approach, since this reduces the number of authored statements for the first persona.) For example, when a belief is objected to, the usefulness of its author decreases and he is therefore led to deepen the argumentation structure on its belief or remove it.

[6] describes a "Knowledge Web" to which teachers and researchers could add "isolated ideas" and "single explanations" at the right place, and suggests that this Knowledge Web could and should "include the mechanisms for credit assignment, usage tracking and annotation that the Web lacks" (pp. 4-5). [6] did not give hints on what such mechanisms could be. This article gives a basis for them.

6 Conclusion

This article aimed to show that a cbwoKB - and hence a cbwoKB based ontology repository - is technically and socially possible, and - in the long term or when creating a new KB for *general* knowledge sharing purposes - provides more possibilities, with *on the whole* no more costs, than the mainstream approach [14][1] where knowledge creation and re-use involves searching, merging and creating (semi-) independent (relatively small) ontologies. However, research on these two approaches are complementary: i) results on knowledge extraction or merging may ease the creation of a cbwoKB, ii) the results of applying these techniques with a cbwoKB as one of the inputs would be better and they would not be lost if stored in a cbwoKB.

This article showed that a cbwoKB can be collaboratively built and evaluated without a selection committee and without forcing the users to discuss or agree on terminology and beliefs. However, to guide users into collaboratively representing knowledge in a normalized and organized way, and hence inserting it "at the right places", other elements are also needed: expressive and normalizing notations,

methodological guidance, a large general ontology, and an initial cbwoKB core for the application domain of the intended cbwoKB. WebKB-2 proposes research results for all these elements. One explored application domain is the "Semantic Web related techniques".

7 References

1. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. International Journal on Semantic Web and Information Systems, 5, vol. 3, pp. 1-22 (2010)
2. Casanovas, P., Casellas, N., Tempich, C., Vrandecic, D., Benjamins, R.: Opjk and diligent: ontology modeling in a distributed environment. Artificial Intelligence Law, 15, vol. 2, pp. 171-186 (2007)
3. Chein, M., Mugnier, M.-L.: Positive nested conceptual graphs. In: ICCS 1997, LNAI 1257, pp. 95-109 (1997)
4. Euzenat, J.: Corporate memory through cooperative creation of knowledge bases and hyper-documents. In KAW 1996, pp. (36)1-18 (1996)
5. Euzenat, J., Mbanefo, O., Sharma, A.: Sharing resources through ontology alignment in a semantic peer-to-peer system. In "Cases on semantic interoperability for information systems integration: practice and applications", pp. 107-126 (2009)
6. Hillis, W.D.: Aristotle (the knowledge web), Edge Foundation, Inc., 138 (2004)
7. Lee, J., Park, J., Park, M., Chung, C., Min, J.: An intelligent query processing for distributed ontologies. Systems and Software, 83, vol. 1, pp. 85-95 (2010)
8. Lewen, H., Supekar, K.S., Noy, N.F., Musen, M.A.: Topic-specific trust and open rating systems: An approach for ontology evaluation. In EON 2006 at WWW 2006.
9. Lowe, D.: Co-operative structuring of information: The representation of reasoning and debate. International Journal of Man-Machine Studies, 23, pp. 97-111 (1985)
10. Martin, Ph., Eboueya, M.: For the ultimate accessibility and re-usability. Chapter 29 (14 pages) of the Handbook of Research on Learning Design and Learning Objects: Issues, Applications and Technologies, IGI Global (2008)
11. Martin Ph.: Exploitation de graphes conceptuels et de documents structurés et hypertextes pour l'acquisition de connaissances et la recherche d'informations. Ph.D. thesis (378 pages), University of Nice - Sophia Antipolis, France (1996)
12. Noy, N.F., Tudorache, T.: Collaborative ontology development on the (semantic) web. In: AAAI Spring Symposium on Semantic Web and Knowledge Engineering (SWKE 2008)
13. Palma, R., Haase, P., Wang, Y., d'Aquin, M.: Propagation models and strategies. Deliverable 1.3.1 of NeOn - Lifecycle Support for Networked Ontologies; NEON EU-IST-2005-027595 (2008)
14. Shadbolt, N., Berners-Lee, T., Hall, W.: The semantic web revisited. IEEE Intelligent Systems, 21, vol. 3, pp. 96-101 (2006)
15. Sowa, J.: Theories, models, reasoning, language, and truth. http://www.jfsowa.com/logic/theories.htm (2005)

Using extended metadata model OMV and metrics in OntoLP Portal

Anderson Bestteti, Larissa A. de Freitas, and Renata Vieira

College Computer, PPGCC,
Ipiranga Avenue, 6681, Partenon, Building 32, Porto Alegre, RS, Brazil
{anderson.bestteti,larissa.freitas,renata.vieira}@pucrs.br
http://www.inf.pucrs.br/~linatural/

Abstract. This paper describes the application of OMV-R, an extension of the OMV metadata model for describing ontologies. The motivation is to help users searching for ontologies and ontology research related resources for reuse. In our extension we included information about ontology evaluation and evolution and we also propose more elaborated ways for describing ontology metrics. The extended model was applied to a web portal for Portuguese ontologies, OntoLP. The model was implemented and evaluated through questionnaires answered by users, researchers and ontology engineers.

Key words: Metamodel, Metrics, Ontology Repository, Ontology Reuse

1 Introduction

One of the main features of ontologies is to enable humans and machines to communicate through semantic formalization. With this purpose, ontologies are used in many areas of Computer Science, such as artificial intelligence, database and software engineering. It has been also the key component for information systems in a increasing variety of domains, well-known examples are the areas of biology, geography and law.

The Semantic Web Project contributed for popularization of ontology as an artifact to build the knowledge representation, shared and reused. With the reuse of ontologies costs can be reduced, mainly in development phase.

As presented in [9], the reuse of ontologies can be seen from two main points: one is to create, extend, specialize and adapt an ontology to build a new one; the other is to combine different ontologies in a single one.

A first problem for reuse is ontology search. Usually, repositories offer poor navigation interfaces [2]. The lack of good quality documentation about available ontologies also contributes to this difficulty. The Metadata models that describe ontologies have been proposed to deal with these problems[6]. The use of a standard vocabulary for ontologies is understood as a basic requisite for a good description of ontologies on the Web.

In this paper we extend and evaluate the previously proposed OMV metadata model through its application to a portal for Portuguese ontologies. The

motivation is to help users searching for ontologies and ontology research related resources.

The paper is organized as follows: in Section 2 we present our extension which we call OMV-R; in Section 3 we describe a set of ontology metrics that we use in the description of the ontologies in our portal; in Section 4 we discuss the evaluation of the model; finally, in Section 5 we show the conclusion and future works.

2 The extended metadata model OMV-R

Our extension of the OMV model and its application to a portal aims to assist users in the process of finding out ontologies for reuse.

The original model proposed by [8] does not include information about ontology evaluation and evolution. We included those elements in the model and we also propose more elaborated ways for describing ontology metrics.

According to [13], metrics are useful to help evaluate ontologies during the build and application phase, enabling a quick and simple understanding about what are being modeled through these structures and facilitating the control of its future evolution. This also holds for ontology reuse.

In [10] a study is performed about the maturity of the current process, methodologies and tools focused on ontology reuse. Their strengths and requirements are identified.

Specifically on the evaluation of ontologies, [16] presents a methodology called Requirements-Oriented Methodology for Evaluation Ontologies (ROMEO). This methodology propose a set of criteria for ontology evaluation. Inspired by these previous works, we considered that ontology evaluation is another important feature in the ontology selection process.

As OMV does not provide such information so important to enable reuse, we extended it with new classes as shown in Figure 1.

2.1 Description of the new classes, properties and relations

This section will present the classes, properties and relations such as were added to metadata model OMV.

OntologyEvaluation Class: this class represents the evaluation of an ontology. The properties of this class are:

- evaluationComments: free text that can be used to comment the evaluation criteria;
- evaluationDate: date when the evaluation was made;
- evaluationValue: value assigned to evaluation criteria.

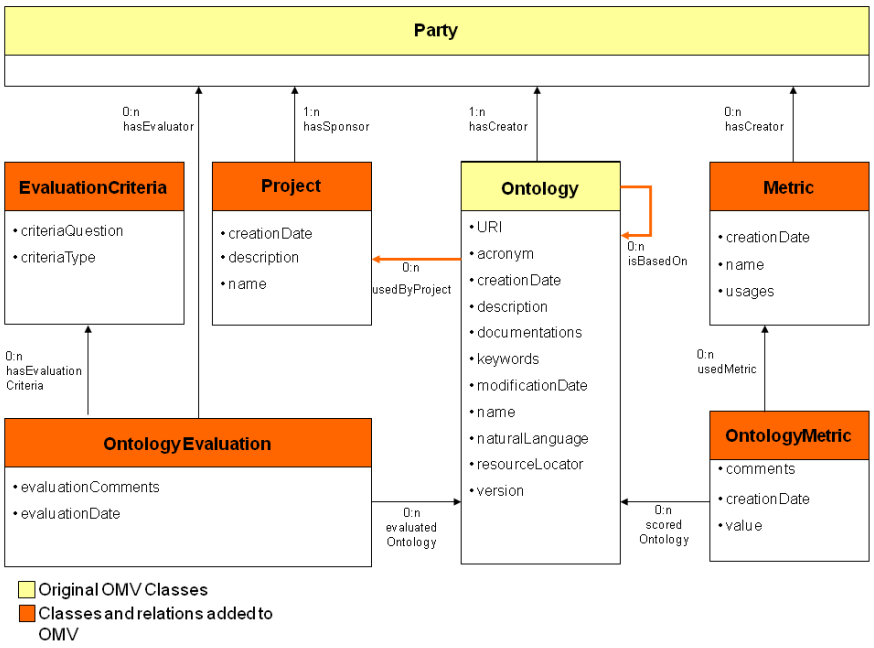


Fig. 1. Extended OMV metadata model .

EvaluationCriteria Class: this class represents the evaluation criteria used to evaluate the ontology. In our implemented model, this class has an evaluation criteria set as discussed in [16]. The properties of this class are:

- criteriaQuestion: question describing the evaluation criteria;
- criteriaType: define whether a question accept a textual or numeric answer.

Metric Class: this class represents metrics. Such metrics can be used to score an ontology, assigning a quantitative value, in order to assist users to select ontologies. In OMV-R there are 20 implemented metrics, which will be discussed in the next subsection. The properties of this class are:

- name: metric name;
- usages: free text to describe and guide the user how to use the metric as well how the metric results may be interpreted.

OntologyMetric Class: this class represents instances of the calculated metrics of an ontology. One or more metrics may be assigned to an ontology, they can help users to select ontologies on the basis of different points of view. The properties of this class are:

- comments: free text to give more information about a metric;
- creationDate: date when the metric was calculated;
- value: value assigned to a metric.

Project Class: this class represents research or commercial projects that use a given ontology or a set of them. The properties of this class are:

- creationDate: date of the project's beginning;
- description: free text to describe the project's objectives and artifacts produced by them;
- name: the project's name.

Relations: here we present the relations among these new classes.

- evaluatedOntology: relation between ontologies with their respective ratings;
- isBasedOn: reference to another ontology which was used in its construction, this reference allows to identify the reuse level that one ontology has, based on the quantity of references;
- hasEvaluator: reference to a research group or enterprise responsible by ontology evaluation;
- evaluationCriteria: reference to an evaluation criteria associated to an ontology;
- hasSponsor: reference to the institution sponsoring the project;
- usedByProject: reference to the project that uses an ontology;
- hasCreator: reference to the research group or institution that has developed the metric;
- usedMetric: reference to the metric assigned to a given ontology.

From the adaptation made, the OMV-R supports now the ontology evaluation description according to the methodologies proposed by [10], [16] and [13]. Another contribution is to allow OMV-R to store new metrics whenever is needed. These metrics may be used to evaluate an ontology over different points of view. Finally, it is expected that the extended model OMV-R provides a greater flexibility in the search of ontologies, as it offers to users additional information about reuse.

In the next section we described the set of implemented metrics.

3 Implemented Ontology Metrics

The model has been extended to accommodate a more elaborated description through metrics. In our application of the model we have implemented ways of calculating a set of metrics to describe the ontologies in the repository. The choice of metrics were based on a detailed systematic review of the literature of ontology metrics. Following the systematic review their presentation was standardized in a common representation formalism, where C is used for classes, CS is used for set classes with subclasses, CR is used for set of root classes, CL is used for set of leaf classes, $H^C(CR_j, CL_i)$ for hierarchy of classes where CR_j is a subclass of CL_i, *prop* for the function that relates the classes in a non-hierarchy and *att* for the function that relates classes with literal values (*string*), I for instances and $inst(C)$ for the function of instantiated classes [13] [12].

Besides, we use $Count$ for count, $SuperClass$ for superclass function, $SubClass$ for subclass function, Max for maximum ($MaxDepth$ and $MaxWidth$), $Depth$ for depth, $Width$ for width, Avg for average ($AvgDepth$, $AvgMaxAvg$, $AvgWidth$, $AvgAttClass$, $AvgPropClass$, $AvgPClass$, $AvgCRC$, $AvgCRCL$, $AvgCIC$, $AvgIC$) and $StdDev$ for standard deviation ($StdDevIC$).

All chosen metrics presented result in numeric values and their calculation could be implemented. The metrics were separated in three groups as follows:

1. Group 1 is simply the count of the basic elements of an ontology (classes, properties, and instances), metrics 1 to 7.

   Number of root classes [15] [1]

   $$Count(CR) = |CR| \tag{1}$$

   where: $|CR|$ is the set cardinality

   Number of leaf classes [15] [1] [14]

   $$Count(CL) = |CL| \tag{2}$$

   where: $|CL|$ is the set cardinality

   Number of classes [7] [1] [14]

   $$Count(C) = |C| \tag{3}$$

   where: $|C|$ is the set cardinality

   Number of properties (attributes) [1]

   $$Count(att) = |att| \tag{4}$$

   where: $|att|$ is the set cardinality

   Number of properties (relations) [1]

   $$Count(prop) = |prop| \tag{5}$$

   where: $|prop|$ is the set cardinality

   Number of properties [7] [1] [14]

   $$Count(att) + Count(prop) \tag{6}$$

   Number of instances

   $$Count(I) = |I| \tag{7}$$

   where: $|I|$ is the set cardinality

2. Group 2 represents the depth and width of the structure, metrics 8 to 12.

Maximum depth [7] [1] [14]

$$MaxDepth(H^C) = \text{the major ontology depth counted from each root class } (CR_j) \tag{8}$$

Average depth [1] [4] [15]

$$AvgDepth(H^C) = \text{the average ontology depth added from all root classes } (CR) \text{ by counted from each leaf class } (CL_i) \tag{9}$$

Average maximum depth to average depth [14]

$$AvgMaxAvg(H^C) = MaxDepth(H^C)/AvgDepth(H^C) \tag{10}$$

Maximum width [1]

$$MaxWidth(H^C) = \text{the major ontology width counted from each class with subclass } (CS_k) \tag{11}$$

Average width [1]

$$AvgWidth(H^C) = \text{the average ontology width added from all classes with subclasses } (CS) \text{ by counted from each class with subclass } (CS_k) \tag{12}$$

3. Group 3 represents the calculation of simple averages and standard deviation from combinations of the basic elements, metrics 13 to 20.

Average number of properties (attributes) to total number of classes [1]

$$AvgAttClass = Count(att)/Count(C) \tag{13}$$

Average number of properties (relations) to total number of classes [1] [14]

$$AvgPropClass = Count(prop)/Count(C) \tag{14}$$

Average number of properties to total number of classes [1]

$$AvgPClass = (Count(att) + Count(prop))/Count(C) \quad (15)$$

Average number of leaf classes to total number of classes [1] [14] [13]

$$AvgCRC = Count(CR)/Count(C) \quad (16)$$

Average number of leaf classes to number of classes root [14]

$$AvgCRCL = Count(CR)/Count(CL) \quad (17)$$

Average number of classes populated to total number of classes [13]

$$AvgCIC(H^C) = Count(inst(C))/Count(C) \quad (18)$$

Average of the total number of instances to total number of classes

$$AvgIC(H^C) = Count(I)/Count(C) \quad (19)$$

Standard deviation of the total number of instances to total number of classes [13]

$$StdDevIC(H^C) = \sqrt{\sum(Count(inst(C)) - AvgIC(H^C))^2/Count(I)} \quad (20)$$

The $AvgIC(H^C)$ and $Count(I)$ were calculated and used in $StdDevIC(H^C)$. It is not directly referenced in papers obtained in the systematic review.

Note that OMV documentation refer to four metrics only: number of class, number of property, number of individuals and number of axioms.

4 Application evaluation

OntoLP Portal[1] offers ontologies and ontology research related resources, with special preference to ontologies written in Portuguese language. At *Home* link the objectives of the portal are shown, whilst the link *Resources* list ontologies and related works. The link *About* contains information referring to the research group involved in this project. Through the *Contact* link others research groups in the field can send their works and suggestions. Finally, the link *Links* shows researches groups that have been colaborated directly or indirectly with the OntoLP Portal as well as events related with ontology research.

An ontology search was performed just to build the repository. We conducted searches in Google [2] specifying the type of query (e.g. "field filetype: owl"), in Swoogle [3], in OntoSelect [4] and pages of projects and research groups working with the subject. Also we announce the OntoLP portal in several Brazilian

[1] http://www.inf.pucrs.br/~ ontolp/index.php
[2] http://www.google.com.br/
[3] http://swoogle . umbc.edu /
[4] http://olp.dfki.de/ontoselect/

research lists and organized an event with the intention of receiving these resources.

As a result we find resources in different domains such as ecology, nanotechnology, art, curriculum, emotion, privacy, network of scientific knowledge, smartphones, music and stimulus equivalence. Today the OntoLP portal has 25 ontologies. Note that the currently portal does not use any advanced structure, such as a metamodel, to store the descriptions of ontologies. Such descriptions were manually built and hard coded in HTML, which are stored in the file system on the Web server. In order to improve the current services of OntoLP portal, a new prototype application was developed that uses the OMV-R to keep the descriptions of ontologies. The next subsection will take a look deeper how this new version of OntoLP portal was developed.

4.1 Infrastructure

A tool has been developed in order to automate the receiving process. This tool reads the ontology and includes information about ontology metrics. It has also another interface for the inclusion of other metadata, to be used in the description of ontologies in the OntoLP Portal, that corresponds with instances of OMV-R. Ontologies are thus received through a submission form. Figure 2 shows the flow of information during the ontology reception process.

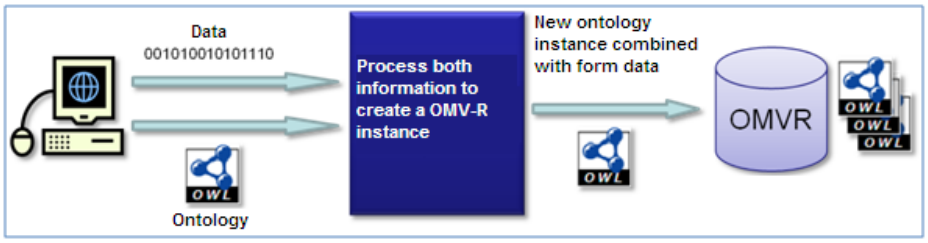


Fig. 2. Flow to store an ontology into OntoLP Portal.

Besides the ability to create OMV-R's instances, the tool offers others services such as information retrieval of the ontologies, ability to create and retrieve ontology evaluation instances as well as the metrics values for each OMV-R instance that represents a description of the ontology. The recovery of information from ontologies is made through a set of methods that read data from OMV-R and then, record the information on a XML file wich will be used by presentation layer of the prototype application. Therefore, this XML file works like a communication area between OMV-R ontology and end user interface.

Finally, the tool uses the Jena's SPARQL query engine to implement the advanced search service in the OntoLP Portal. The query is performed on the OMV-R, specifically over the Ontology class to retrieve all the ontologies that match the user's filter.

Figure 3 presents the OMV-R metric instances for the stored ontologies. Notice that the OntoLP Portal prototype application can be accessed through following link: http://www.ontolp.com.br. Both versions of portals are available for readers to discover the differences between portals (take a look at previous page on footnote for the URL of the current portal).

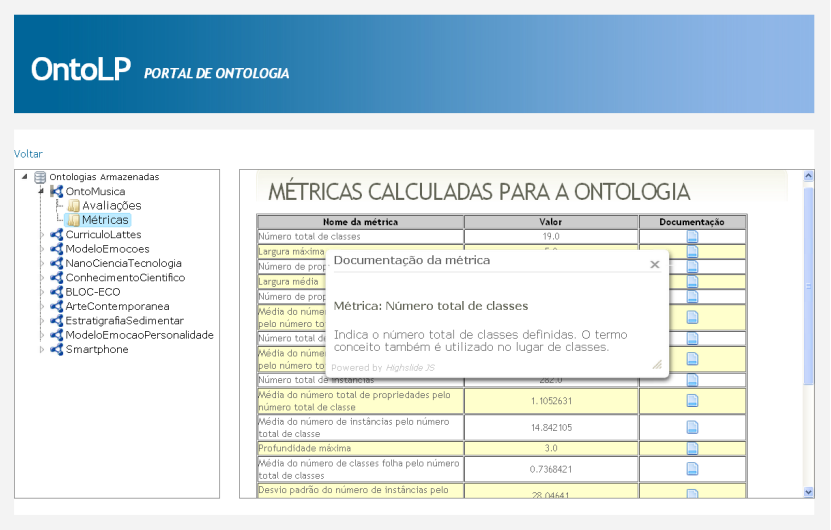


Fig. 3. Portal OntoLP interface showing an instance of OMV-R.

4.2 Evaluation

We applied two questionnaires to evaluate the adaptations made to the portal which follows the new proposed model.

The goal of the first questionnaire was to evaluate the opinion of a users' group, researchers and engineering of ontologies about the usefulness of metrics in specific repositories, in this case in OntoLP portal.

In the first phase, the questionnaire (in print format) was applied to people in national workshop about ontologies. In the final phase the questionnaire (in digital format) was applied to visitors of the OntoLP portal. The questionnaire was available for a month (December 6, 2009 until January 6, 2010).

The final result showed that 76 % respondents have developed ontologies, 83 % have searched for ontologies written in English, Portuguese, or Spanish. The language most cited was English and Portuguese was in the second position. The sample consisted of 14 PhD, 11 MSc and 5 senior experts. At last, ontologies in many domains were mentioned.

We observed that there was no preference regarding groups 1, 2 and 3, their usefulness was better appreciation as a whole. Most people consider the use of metrics important for the portal.

The respondents commented about the lack of more specific metrics (total number of relations' types, where such relations would be " is-a", " part-of") and about the lack of the presentation in percentages the results (percentage of classes that did not have superclasses). These suggestions will be considered as future work.

The goal of the second questionnaire was to get impressions about the new services of the OntoLP Portal such as description and retrieval of ontologies based on the OMV-R metadata.

Here, the sample was formed by 19 respondents. They were instructed to visit the OntoLP Portal and to evaluate the new services such as ontology submission and search. Next, the respondents were oriented to compare the services of the current version of the portal. When asked whether these new services have eased the ontology search process, about 58% of respondents agreed, 26% remained neutral and about 15% disagreed.

Despite some disagreement about how easy is to locate ontologies in the new OntoLP Portal, about 95% of respondents believe that the new ontology submission and search services are better than the original one.

In addition to the two close questions about the satisfaction and improvement about the ontology search services, the respondents were conducted to comment referring to other information that could be added to the metadata model OMV-R. About 25% of respondents have contributed with suggestions; 32 % said that the new set of the model's information was sufficient or reasonable; and 36 % didn't answer.

The suggestions are related mainly with the ontology domain, and this information is the best way to locate ontologies. Other respondent claimed that the new portal should offer information about the relationship among ontologies, as evidence: "At first glance I would say that at some point we will need a hierarchy of ontologies (when there are more than 20 or 30 ontologies) and some mechanism to relate them (meta-ontology)". Notice tha both questionaries can by accessed through the following links: http://www.inf.pucrs.br/~ontolp/questionario.php (for metric's questions) and http://www.ontolp.com.br/questionarioOMVR.php (for metamodel and prototype application's questions)

5 Conclusion and Future Work

This paper presented an extended model based on OMV, which is called OMV-R. The new model includes key ontology metrics identified through literature review. The model was applied to OntoLP portal which aims to maintain and distribute Portuguese resources. The model is used for the description of the resources, and was also useful as a basis for new search and retrieval services. Finally, this paper discussed the surveys performed to evaluate the model, the new portal and the use of metrics, where the objective was to assess with the

community the benefits and contributions of the model for ontology description and location as well as the process of reuse of such resources.

We hope to have contributed with ideas for describing ontologies that can be included in previously proposed standards. We have also shown its usefulness for the development of software that can help researchers and engineers to find and inspect available ontologies in an organized and efficient way.

Potential future work are: (*i*) adaptation of the OMV-R API to work with the OMV-R model stored in commercial database as well as free ones; (*ii*) the development of Web services for distributed applications; (*iii*) improving the advanced search interface of the new OntoLP Portal, in order to allow users query any property of the metamodel through free-text. Nowadays only three properties have been added to advanced search interface; (*iv*) building mechanisms for visualization of ontologies described in the metadata model OMV-R.

References

1. Cross, V., Pal A.: Metrics for ontologies. In: Annual Meeting of the North American Fuzzy Information Processing Society, pp. 448-453, (2005).
2. Ding, L. Finin, T. Joshi, A. Pan, R. Cost, R. S. Peng, Y. Reddivari, P. Doshi, V. Sachs, J.: Swoogle: A Semantic Web Search and Metadata Engine. In: Proceedings of the thirteenth ACM international conference on Information and knowledge management, pp. 652-659, (2004).
3. Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D., Patel-Schneider, P.: OIL: An Ontology Infrastructure for the Semantic Web. IEEE Intelligent Systems, v. 16, pp. 38-45, Mar./Apr., (2001).
4. Gangemi, A., Catenacci, C., Ciaramita, M., Lehmann, J.: Modelling ontology evaluation and validation. In: The Semantic Web: Research and Applications, pp. 140-154, (2006).
5. Gruber, T. R.: Toward principles for the design of ontologies used for knowledge sharing. International Journal of Human Computer Studies 43, 907-928 (1995).
6. Hartmann, J.; Bontas, E. P.; Palma, R.; Gmez Prez, A.:Demo - Design Environment for Metadata Ontologies. In: The Semantic Web: Research and Applications, pp. 427-441, (2006).
7. Lozano-Tello, A., Gmez-Prez, A.: Ontometric: A method to choose the appropriate ontology. Journal of Database Management 15, 1-18 (2004).
8. Palma, R.; Hartmann, J.; Haase, P.: OMV Report 2.4 - Ontology Metadata Vocabulary for the Semantic Web. Technical Report, p. 94, (2008).
9. Pinto,H. S, Martins J. P.: Reusing Ontologies. In: AAAI 2000 Spring Symposium Series, Workshop on Bringing Knowledge to Business Processes, pp. 77-84, (2000).
10. Simperl, E.: Reusing Ontologies on the Semantic Web: A Feasibility Study. Data and Knowledge Engineering, v. 68-10, pp. 905-925, October, (2009).
11. Smith,M. K., Welty, C. , McGuinness,D. L.: OWL Web Ontology Language Guide: W3C Recommendation 10 February 2004.
12. Tartir, S., Arpinar, I.B., Moore, M., Sheth, A. P., Aleman-Meza, B.: OntoQA: Metric-based ontology quality analysis?. In: IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources, p. 9, (2005).

13. Tartir, S., Arpinar, I.B.: Ontology Evaluation and Ranking using OntoQA. In: First IEEE International Conference on Semantic Computing, pp. 185-192, (2007).
14. Yang, Z., Zhang, D., Ye, C.: Evaluation Metrics for Ontology Complexity and Evolution Analysis. In: IEEE International Conference on e-Business Engineering, pp. 162-170, (2006).
15. Yao, H., Orme, A.M., Etzkorn, L.: Cohesion metrics for ontology design and application. Journal of Computer Science 1, 107-113 (2005).
16. Yu, J. Thom, J. A. Tam, A.: Requirements-Oriented Methodology for Evaluating Ontologies. Information Systems, v. 34-8, pp. 766-791, December, (2009).

Ontology Repository for User Interaction

Martins Zviedris[1]

[1] Institute of Mathematics and Computer Science, University of Latvia, Raina bulv. 29, Riga LV-1459, Latvia
Martins.Zviedris@Lumii.lv

Abstract. The systematization and the interaction with ontologies are problems that deserve more attention. One of the key aspects is that ontology repositories should also work as the first step in a data interaction process for end-users, not only as a collection of ontology schemas. We propose a novel systematization of similar domain ontologies described by a high-level abstraction domain ontology that could be used as a domain ontology repository and access point to gather instance data.

Keywords: Ontology Systematization, Ontology Interaction

1 Introduction

Data systematization, representation and accessibility are key factors for the data usage. In the Semantic Web, the state of art for the data representation is ontology and currently ontology repositories are used to store collections of ontology schemas. Main disadvantage is that ontology repositories only allow to access ontology schema. Thus, this leads to an effect that ontologies are developed in an isolated way, as there is not provided access to the real data that would motivate to interlink them.

First, we propose that an ontology repository should contain link to instance data. Second we propose that it would be better to organize a repository around specific domain and group domain ontologies by additional domain specific meta-information. Added meta-information should be organized by a high-level abstraction domain ontology, described in more detailed in section 3. Thus, it would be easier to find similar onotologies and define ways to merge them together. Third, domain repositories would also work as a first step in data interaction for a domain expert user or an intelligent agent. The user would use the repository to select ontologies that contain the data of interest and use selected ontologies to construct relevant data queries. The repository thus becomes a bridge towards the real data.

2 Practical Experience and Proposed Solution

In practice we have encountered a problem, where we had to develop an ontology for medical researchers that describes data from different disease registries [1]. Since

instance data was originally stored in SQL databases then we had to work with SQL-like ontologies; we will preset simple ontology examples below. The role of ontologies was to integrate eleven disease registries and to allow medical researchers to use ontologies as access point to the real data. Thus there was no need for elaborated ontology mechanisms. The only goal was to use ontologies to enable medical researchers to sort through vast amounts of data from registries without programmer's assistance.

A naïve approach is to merge all the registries into single, large ontology that could be stored in the data access point. We have successfully implemented the naïve approach. However, in this case medical researchers hardly comprehend the resulting ultra-complex ontology. The naïve approach needs improvement, as understandability because of ontology is crucial for medical researchers to use the ontology for actual data selection.

Based on the medical domain we will describe a more elaborate solution. The solution involves interaction of two steps. In the first step, a medical researcher has to choose registries that interest him. In the second step, he has to select and obtain data from selected registries. As each registry can also be perceived as an ontology then we need to develop a disease domain repository from where the medical researcher selects ontologies that interests him. So, in the first step the medical researcher selects registries (ontologies) from high-level abstraction disease ontology that are merged into single ontology. In the second step, the end-user can interact with data better as ontology consists of a smaller number of classes and most of them are of end-user's interest. The main idea in this approach is to develop a high abstraction level ontology that represents features from all registries. As a result, the end-user in the first step can select registries (ontologies) more conveniently and can further work with single ontology containing the needed data.

3 A Detailed Example of Medical Domain

We start with a requirement to integrate different medical disease registries into single integrated registry. Each registry contains patient's data. We propose that all registries should be stored in a medical domain repository. Also, it is required that ontology records contain links to instance data. As for medical researchers it is often necessary to work with several ontologies at a time then we also propose that these ontologies need to be organized in a more elaborate way using a high-level abstraction medical disease registries ontology.

To better understand a high-level abstraction ontology, we will build it from medical domain examples. We need to integrate two simplified registries – diabetes registry depicted in Fig 1. and cancer registry depicted in Fig 2. In practice these registries contained also other information connected to simplified solution classes and consisted of about 10 classes and about 20 enumerated classes used for classification.

By analyzing depicted registries we can identify similar structure in them. Each registry contains such general concepts as person, disease information, disease details and disease cure. A question arises whether these similarities can be used to develop a

high-level abstraction ontology. As we need to develop a high-level ontology that is used in a repository for ontology selection then we need to consider only those concepts that can ease selection process. As concept "person" does not contain information useful for ontology selection it should not be considered for a high-level ontology. Still, it could be useful to mark person concept as a concept that can be used to merge registries, thus, ontologies merging could be done at least semi-automatic.

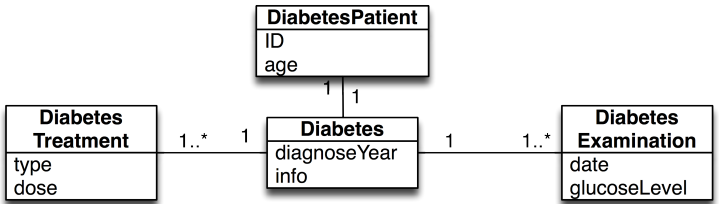


Fig. 1 Simplified diabetes registry

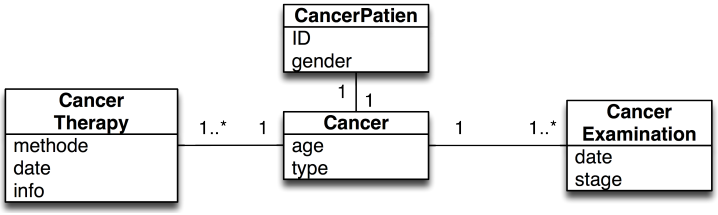


Fig. 2 Simplified cancer registry

We can identify a pattern that can be described as a disease has a treatment and an examination. The pattern is general and can be consider as a high-level abstraction ontology. The pattern can be depicted as ontology in Fig. 3. (additional information about the treatment and the examination may be added).

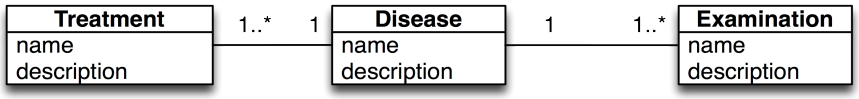


Fig. 3 Common disease ontology

Such pattern was discovered in eleven registry ontologies that ware developed for medical registries. We can see that the pattern ontology is very simple and easy to grasp for the end-user. In addition, most medical disease ontologies can be described as an instance of this ontology. Still, this pattern ontology lacks meta information about actual registries and where one can find instance data and thus link to it. Also, it could be possible that data is stored in more than one place; for example, each clinic could have own cancer registry and specific disease details. We add links between the high-level abstraction ontology instances and corresponding ontologies. Links are also added towards real data and contain information about, for example, how long

data is gathered, thus links allows additional selection possibilities. Schematically structure is depicted in Fig 4.

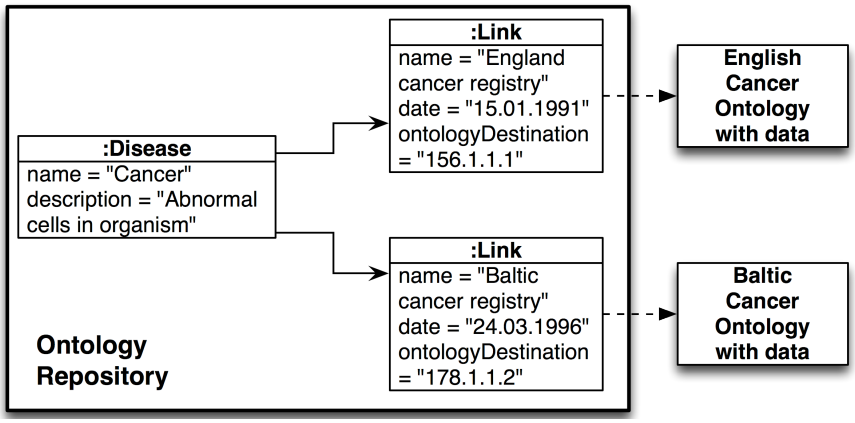


Fig. 4 Connection between ontology levels

We will sketch how a medical researcher could gather relevant data about cancer and its treatment possibilities using the proposed solution. We will not describe a specific way to query ontology as it can be done through SPARQL or more preferably by a graphical query language [2, 3]. Firstly, a medical researcher connects to a disease domain repository and queries high-level ontology depicted in Fig. 3. He restricts that he is interested in disease with name = “Cancer” and all corresponding treatments. As a result he gets instance pairs of cancer and corresponding cancer treatment and link information to ontologies that contain instance data. At this point the medical researcher can further restrict registries that interest him, for example, he could be interested only in registries that gather data at least 10 years. If he selects at least two registries, for example, the Baltic cancer registry and the England cancer registry then ontologies from both registries are merged into single ontology. This can be achieved using information that both ontologies contain similar concepts “person” and that in both ontologies is present abstract Cancer class as super class of specific Cancer classes that contain data. This ontology is presented to a medical researcher, where he can gathers clinical data for further analysis.

4 Related Work

From technical point of view, OWL2.0 allow to use punning, where an object can be represented as a class. Still, metadata addition to ontology does not solve the problem of how to group similar domain ontologies together for further interaction. Most existing ontolory repositories just collect ontologies and allow to reuse ontologies. For example [4] does not give possibility for further interaction with collected ontology data that is needed for common users. They even do not collect links to existing data.

5 Results and Future Work

In practice, we have designed and implemented the naïve approach with six different disease registries [1] integrated into single ontology. To allow medical researchers query data, we have developed and implemented a graphical query language [2]. After implementing the prototype we gathered user feedback to evaluate our work. Most valuable feedback that we got was that ontology was too ultra-complex for medical researchers. Also, it was relatively easy to produce queries in ontology part, with witch a researcher was familiar with. Other valuable feedback was that medical researchers are interested in registries meta information, for example, how long data has been gathered.

It would be important to practically implement and test specific domain repository with access to ontologies that contain real data. The proposed approach needs to be developed in more details to interlink between an ontology repository and ontologies that contain real data. Interesting problem also would be to find whether there should be predefined ontologies for specific diseases that could be configurable for each registry that contains such disease data and such ontology could be used for mush-up. As we have only developed the theoretical approach for medical domain it is important to go further into other domains and see possibility of a high-level abstraction ontology approach. We should mention that such domain repositories would be useful for intelligent agents as they could find links to similar data in one place.

Acknowledgements

I would like to thank prof. Guntis Barzdins for valuable discussions and prof. Karlis Podnieks for useful ideas. Also I would like to thank Arturs Sprogis and Renars Liepins for valuable assistance.

References

1. Barzdins G., Liepins E., Veilande M., Zviedris M., Ontology Enabled Graphical Database Query Tool for End-Users, Selected papers from DB&IS'2008, Hele-Mai Haav (Eds.), Frontiesrs in Artificial Intelligence and Applicatons series, IOS Press, 2009. 187:105--116
2. Barzdins, G., Rikacovs, R., Zviedris, M.: Graphical Query Language as SPARQL Frontend. In Grundspenkis, J., Kirikova, et. al. (Eds.), Local Proceedings of 13th East-European Conference (ADBIS 2009), pp. 93--107. Riga Technical University, Riga. (2009)
3. Chen, H., Wang, Y., Wang, H., Mao, Y., Tang, J., Zhou, C., Yin, A., Wu, Z.: Towards a semantic web of relational databases: A practical semantic toolkit and an in-use case from traditional chinese medicine. In Cruz, I.F., et.al, eds.: 5th International Semantic Web Conference. LNCS, vol. 4273, pp. 750--763. Springer (2006)
4. N. F. Noy, N. H. Shah, P. L. Whetzel, et. al. BioPortal: ontologies and integrated data resources at the click of a mouse. Nucleic Acids Research, 2009

Evaluation Framework for Ontology Development and Management Methodologies

Dionisis D. Kehagias, Dionysia Kontotasiou, and Dimitrios Tzovaras

Centre for Research and Technology Hellas,
Informatics and Telematics Institute,
57 001, Thermi, Greece,
diok@iti.gr, dkonto@iti.gr, dimitrios.tzovaras@iti.gr

Abstract. Based on the IEEE 1074-1995 standard for software development process we define a set of complement criteria for the evaluation of existing ontology development and management methodologies. We use this evaluation framework in order to benchmark the most well known approaches for developing ontologies from scratch, as well as reusing ontologies that are stored in ontology repositories. The result of the evaluation process is to identify the shortcomings of existing ontology development and management methodologies and validate the use of the aforementioned criteria for the establishment of a generic evaluation framework. Moreover, the conducted evaluation procedure reveals the degree of conformance of the benchmarked methodologies to a set of standardized criteria. In order to include the newly introduced evaluation criteria whose purpose is to support more ontology-specific evaluation aspects, we extend the evaluation framework introduced in the Ontoweb project by also adding nine additional methodologies.

1 Introduction

Ontology development and management are two knowledge engineering processes that are required either for constructing an ontology or an ontology network from scratch or reusing existing ontologies. Even though many visualization support tools that are available today facilitate the various steps of the ontology lifecycle, the core development of an ontology remains a manual task that requires good knowledge of the domain to be modeled, as well as good modeling skills and experience. To this end a deep knowledge of existing methodologies about ontology management and development are necessary for the adoption of the best practices available in the market.

This paper aims at the provision of an in-depth review of the most common methodologies for ontology development and management by introducing a set of evaluation criteria to enable benchmarking of the existing methodologies. The proposed set of criteria is based on the IEEE 1074-1995 standard for software development process [1] that defines the main processes involved in the administration, preparation, development and integration of a software project. Various ontology construction methods for building single ontologies from scratch are presented in [2], where it is noted that none of them is the most adequate for all situations; instead, each one has its own use, depending on the application's specificity. Our work extends the IEEE 1074-1995 in order to form the basis for the development of a concrete methodological approach and

a set of guidelines for ontology authoring which can be used and applied as a set of evaluation criteria for new ontologies as well.

In particular the paper focuses on the introduced criteria by providing a thorough analysis of their impact on existing methodologies. The new evaluation criteria are based on practical observation of ontologies from real repositories and a set of practical guidelines about the establishment of an ontology evaluation and refinement best practice. By extending the IEEE standard we conduct benchmarking of existing ontology development and management methodologies in order to identify potential inefficiencies on existing methodologies. Our evaluation framework extends the one introduced by the Ontoweb project [3] by adding nine additional methodologies. An evaluation process is conducted in order to investigate the degree in which the existing methodologies provide support of the evaluation criteria presented in this paper.

2 The IEEE Software Development Standard

The set of criteria used for the definition of our evaluation framework is based on the IEEE 1074-1995 standard for software development process. According to the IEEE definition [1], software is ”computer programs, procedures, and possibly associated documentation concerned with the operation of a data processing system; e.g. compilers, library routines, manuals, and circuit diagrams”; ontologies are part of software products. In order to construct an ontology similar processes are required, such as design of class hierarchies, development of the ontology, validation of an ontology by executing a reasoner and generation of documentation. Based on these similarities, the proceeses involved to the creation of an ontology from its conceptualization to the development and documentation can be described by the IEEE standard after its adaptation to the specific characteristics of ontology development.

According to this standard any software development process is broken down in processes. We adapt each process to existing ontology development methodologies and propose a set of applied criteria. The purpose of the set of defined criteria is to drive the evaluation procedure for different ontology development and management methodologies. These criteria are intended to stand as complementary elements to the IEEE 1074-1995 standard extending its application to the ontology development and management processes. The IEEE 1074-1995 standard defines the following processes for software.

> *Project management processes*. These processes adhere to the procedure required for setting up a software development project. Their purpose is to ensure the right level of management throughout the entire project life cycle. They include activities related to project initiation (such as participants, scheduling, etc.), project monitoring and control, and quality management. The activities proposed by the IEEE standard for these processes are applicable to any software product and therefore they are recommendable to be applied in ontology development. Ontology management activities include scheduling, control and quality management. The scheduling activity identifies the tasks to be performed, their arrangement, and the time and resources needed for their completion. This activity is essential for ontologies that

use ontologies stored in ontology libraries or for ontologies that require a high level of abstraction and generality. The control activity guarantees that scheduled tasks are completed in the manner intended to be performed. Finally, the quality management activity assures that the quality of each and every product output (ontology, software and documentation) is satisfactory.

Development-oriented processes. This category includes the processes that are used in order to produce, install, operate and maintain the software and retire it from use. They are divided into three groups:

a. *Pre-development processes*. They are performed prior to the actual software development. They involve activities related to the study of the software installation environment, and to feasibility studies. During the ontology predevelopment an environment study identifies the problem to be solved with the ontology, the applications that will consume the ontology, etc. Also during the pre-development, the feasibility study answers questions such as: "what is the purpose for building such and ontology"; "is this ontology suitable to solve the problem for which it is designed?"
b. *Development processes*. These are the required processes for building the software product. They include: requirements, which are comprised of iterative activities directed towards developing the software requirements specification; the design process, the goal of which is to develop a coherent and well-organized representation of the software system that meets the requirements specification; and the implementation process, which transforms the design representation of a software product into an implementation language. Obviously, if ontologies are to be used by computers, they have to be implemented like software products. Thus, firstly in development, the specification activity states why the ontology is being built, what are its intended uses and who are the end-users. The conceptualization activity, like software design process, structures the domain knowledge as meaningful models at the knowledge level either from scratch or by reusing existing models. Finally, the implementation activity builds computable models in an ontology language.
c. *Post-development processes*. They are related to the installation, operation, support, maintenance and retirement of a software product. They are executed after the software construction. As in software, these activities are applied to ontologies in the way we explain in what follows. During post-development maintenance activities concern updates and corrects the ontology if needed. Also during post-development, the ontology is (re)used by other ontologies or applications. Evolution involves managing ontology changes and the impact of updated versions of the ontology, taking into account the applications and the environments on which it can be used.

Integral processes. These processes are required for successful completion of software project activities. They are executed concurrently to the software development-oriented processes and include those activities that are necessary for the successful integration of the overall system. With respect to software development and management they cover the processes of knowledge acquisition, verification and validation, software configuration management, documentation development and training. The activities proposed by the standard for these processes can be applied

also to ontologies in the following ways. Ontology evaluation involves assessment of ontologies and associated execution environments from a technical point of view. Documentation for ontologies is as necessary as in software products. Configuration management can be applied to ontology as a means of assessment to make sure that the developed ontology adheres to its original requirements.

After we have seen how the IEEE Standard can be applied to ontology development, in the next Section we describe the complement criteria that we have introduced to the above processes defined in the IEEE standard in order to establish an ontology management evaluation framework.

3 Evaluation Criteria

We present here the additional ontology evaluation criteria that we have defined with respect to the aforementioned IEEE standard in order to make the standard applicable to the ontology development and management processes. For presentation purposes we introduce the new criteria based on practical experience by the application of the aforementioned process categories that are defined in the IEEE standard on existing ontology repositories. Such a repository on which we have been experimented is the ORATE (Ontology Repository of Assistive Technologies), which is located at `http://ontologies.informatik.uni-bremen.de/`. ORATE hosts a large collection of assistive-related ontologies.

3.1 Project Management Processes

Project management includes on-going activities that are executed during the whole period of the ontology capture and development process. We do not provide new criteria for this category. However we review the most common methodologies for ontology management processes in general with respect to their support for project management activities.

3.2 Pre-Development Processes

During the ontology pre-development phase an environment study identifies the problem to be solved with the ontology, the applications where the ontology will be integrated, etc. Also the feasibility study provides a mechanism to refine the vision statement and to find out whether an ontology is actually worthwhile in terms of expected costs and benefits. A feasibility report not only provides recommendations of how to re-fine the vision statement, but also the material and rationale underpinning it. As previously, we do not provide new criteria for this category. However we review the most common methodologies according to their support for these pre-development processes.

3.3 Development Processes

The development processes category includes most of the new criteria. Firstly, in this process category that involves those processes that are executed in order to prepare the main development process, we introduce two new criteria, namely *concept hierarchy* and *property structure*. Hierarchy renders a key aspect in the ontology development process. Since hierarchy of the various concepts is part of the ontology design process and involves decisions made at the initialization of the ontology development phase it is related to development processes. More specifically it is related to ontology conceptualization. Evaluation metrics that are derived from this criterion are the size, the depth, and the breadth of hierarchy, the density (average branching of concepts), etc., which define the overall complexity of the ontology. A flat concept hierarchy for example usually means that there are too many concepts on the same level [4]. This phenomenon implies the existence of unexploited grouping possibilities for concepts of similar kinds, e.g. to be grouped together under one more general concept. Another example is the existence of branches very differently structured than others (e.g. very big depth), something that results in an unbalanced taxonomy. In general, if the level of abstraction to which the concepts refer is not taken into careful consideration, the result will be an inappropriate design of the ontology.

Next criterion is the *property structure*. This criterion, like the previous one, refers to development processes, as the definition of an appropriate property structure is realized in the development of an ontology. This criterion is associated with metrics such as the size, the depth/breadth of hierarchy, density and complexity of the ontology, etc. It is often observed in ontologies for which data or object properties are not properly structured or not structured at all. In this case, a restructuring process might be necessary by exploiting grouping possibilities for properties of equal domains/ranges or their functions.

Two more criteria in the same category are *domain/range definition of properties* and *disjointness restrictions*. The domain/range definition criterion covers the activity of defining the environment to which the ontology has impact. For this reason and based on the IEEE standard we classify this as a development process. The existence of properties which do not define their domain/range can cause significant inconsistencies when using the ontology. Another common case is the existence of object properties which do not define their range, but instead they appear in restrictions of concepts, in which the range is set (as a condition of the concept).

Next criterion, disjointness restrictions [5] is also implemented as a development process. Its impact is visible when the ontology is used as part of an overall application (e.g. when instances are added, forms are created, queries have to be managed, etc.). Although most concepts inside the ontology are usually pairwise disjoint with each other, this condition is not always there. On the other hand, for some other concepts disjointness should not hold when there exists an individual that is an instance of two classes. In this case disjointness restriction should be removed from the two classes.

3.4 Post-Development Processes

Three ontology-specific criteria are introduced in the post-development process category, that correspond to activities such as support, maintenance and retirement. These

are *repetition of similar ontological concepts*, *subtraction of modules* and *naming conventions*.

The first criterion is presented in the post-development processes category because it is related to the main ontology post-development activities of maintenance and reuse. In particular this criterion concerns modularization (e.g. what modules are defined in the ontology, how they are defined, if they can be imported/exported/reused, etc). If similar ontological concepts are repeated frequently throughout the ontology structure, they can possibly be combined to one module and reused whenever necessary. Hence, repeated concepts can be defined only once and their use be extended within other definitions.

The second criterion, subtraction of modules, is closely related to the previous one, since it refers to subtracting modules in general (either functional or logical) from the whole ontology, which is also the result of applying measures in order to eliminate repetition of similar concepts. Such an action can reduce the overall complexity and elucidate dependencies between various ontology parts. From our practical experience with ORATE ontologies we noticed that, some ontologies have duplicate definitions of the same concept or concepts which are very similar (or almost identical to each other). In such a case, it is necessary to eliminate duplicate definitions and remove similar concepts or merge them to a single one. Moreover, there might be properties initially created for some purpose, but finally never used at all. These properties should also be removed. All these steps usually result in an ontology of reduced complexity, more ”clear”, compact and readable.

Moreover, in this category we have included a criterion about naming conventions [6]. This criterion has to do with the formulation of ”good” terms and definitions, where essential features should be satisfied by all naming conventions (e.g. nominal, verbal, etc). Circularity in definitions should be avoided and junk categories should be eliminated.

3.5 Integral Processes

In the group of integral processes we have included a criterion about documentation and information visualization. The integral processes include the activities of validation and documentation development. They are related to documentation, syntax (syntactic correctness, breadth of syntax used), and governance in used terms, etc. The specific criterion concerns the activity of enriching the ontology with additional information (e.g. natural language comments/annotations, metadata, implementation code, etc.), as well as the collection of documents and explanatory comments generated during the entire ontology building process. In general, this issue has to do with anything that could be useful to help users, who did not participate in the ontology development, to understand and learn how the ontology was built.

4 Evaluation of Existing Ontology Development Methodologies

This section presents a survey of ontology development methodologies and the results obtained after conducting benchmarking of the existing tools with respect to our IEEE

1074-1995 standard-based evaluation framework. A short description of the main characteristics of the ontology development and management methodologies that participated in our benchmarking evaluation is provided in what follows.

4.1 Ontology Development and Management Methodologies

The *Cyc* methodology which arose from experience of the development of the *Cyc* knowledge base contains a huge amount of common sense knowledge [5]. After evaluating *Cyc* we saw that it provides limited description of the criteria and processes described in Section 3. For example, the criteria for formulating the concept hierarchy are not mentioned.

The *Uschold and King* ontology development method [7], also known as the "skeletal method", is based on the experience of developing the Enterprise Ontology which is a collection of terms and definitions relevant to business enterprises. This method is composed of four distinct stages: identification, construction, evaluation and documentation. However, some criteria are missing in the processes it does propose (development and integral), particularly: concept hierarchy, property structure, naming conventions and information visualization.

The *Toronto Virtual Enterprise Method* (TOVE) [8] was derived from the authors' own experience in developing ontologies for business and corporate processes, using motivating scenarios to describe problems and examples that were not addressed by existing ontologies. This methodology is very formal and can be used as a guide to transform informal scenarios in computable models. However, it shows similar omissions as the previous methodology. In particular, no reference is made to the criteria concerning: naming conventions and documentation but it provides more details for the criteria in the processes it does propose, i.e. development and post-development processes. These include concept hierarchy, disjointness restrictions and subtraction of modules.

The *METHONTOLOGY* framework [9] is essentially a descriptive method that provides automated support for ontology development and is based on the IEEE 1074-1995 standard for software development; it suggests which criteria should be accomplished when building ontologies, but it does not provide guidance as to how they should be carried out. Thus, some activities and techniques relevant to the post-development processes should be specified in more detail.

The method based on *SENSUS* [10] is completely different from the others. Domain ontologies built using the *SENSUS* approach share the same high level concepts (or skeleton). Thus systems that use such ontologies will share a common structure of the world, and it would be easier for them to communicate because they share the same underlying knowledge. However, this methodology does not mention at all any post-development processes that are required in order to ensure that the resulted ontologies satisfy a set of usability standards.

On the contrary, *CommonKADS* methodology [11] does not put emphasis on management and integral processes but only to pre-development and development processes. According to this methodology, the phases to ontology design are: feasibility study, refinement and evaluation, while the management processes are missing.

The partially well-documented *On-To-Knowledge* methodology [12] includes the identification of goals that should be achieved by knowledge management tools and it

is based on an analysis of usage scenarios. It is a centralized ontology development method that risks becoming too much geared towards a single application and not towards satisfying general management and development criteria. This could also be a potential problem for *ROD* [13] which is a methodology that is used to build ontologies for under developed domains without ensuring that the resulted ontologies satisfy a set of usability standards. It consists of three processes: domain analysis, document and language processing which correspond to development processes.

Holsapple et al. [14] focus their methodology on the collaborative aspects of ontology engineering but still aim at a static ontology. According to their methodology a knowledge engineer defines an initial ontology which is extended and modified based on the feedback from a panel from domain experts. This feedback does not include criteria relevant to the post-development and integral processes.

The *DOGMA* methodology [15] is quite similar to *DILIGENT* [16] and *HCOME* [17] methodologies. All these efforts move towards the third-generation of ontology engineering methodologies. Specifically, they focus on management and development criteria and thus emphasize on issues concerning good representation and architecture of the ontologies. In addition, these methodologies consider evolving ontologies, pointing on the importance of documentation, versions management and merging of ontologies. All three of them omit the post-development processes criteria.

UPON (Unified Process for ONtology building) [18] is an incremental methodology for building ontologies. This methodology stems its characteristics from the Software Development Unified Process and uses the Unified Modeling Language (UML) to support the preparation of all the blueprints of the ontology project. Because of its nature, *UPON* does not deal with management and integral issues. On the other side it describe in detail the development criteria, which is an advantage over the adoption of other methodologies, that roughly cover the same criteria as *UPON*.

Karapiperis and Apostolou [19] proposed a methodology which complies almost perfectly with our criteria. This approach starts with the deployment of an initial version of the ontology, created by the coordinator, based on the participants' requirements. The initial version is being iteratively evaluated by the participants and it finally evolves into the final version. It ensures that all participants agree and accept the resulting ontology, being a product of a joint team effort. These phases comply also with Holsapple's phases [14]. However, due to the iterative cycles of the consensus building mechanism the collaborative ontology approach require more time and effort to deployment as opposed to other approaches.

In [20] a novel modeling methodology for biomedical ontologies is designed called GM. This methodology has the similar compliance to our evaluation framework as the above methodology. A key feature of this methodology is the use of Concept Maps (graphs consisting of nodes representing concepts, connected by arcs representing the relationships between those nodes) throughout the knowledge acquisition process. Unlike GM, *iCapturer* [21] makes use of all the development criteria except from concept hierarchy and property structure. This methodology does not include the applicability of an ontology in a given application domain.

Last but not least *NeOn* (http://www.neon-project.org) is a framework for developing networked ontologies. It is one of the most comprehensive works in terms of

ontology engineering. The framework incorporates a methodology. The first version of the *NeOn* Methodology for collaboratively building networks of ontologies is available since February 2008. This version of the methodology is focused on the post-development and integral processes. The second version of the NeOn Methodology [22] for collaboratively building networks of ontologies launched in February 2009.

4.2 Benchmarking Results

The result of the comparative evaluation of the aforementioned key methodologies is illustrated in Table 1 where the conformance of each methodology to our evaluation criteria is shown. Rows in Table 1 represent the various methodologies, while columns represent the different groups of processes as they are defined in the IEEE 1074-1995 standard. Each cell in the table can be filled in with five types of values. The value "described" (D) means that the approach establishes for the considered metric: how to do each task, when to do it, who has to do it, etc. The value "proposed but not described" (P-ND) means that the methodology of the corresponding row identifies the process that is written in the column as a process to be performed during the ontology development process but there is no description for this process. The value "not proposed" (NP) means that public documentation does not mention the non-considered aspect. Finally "limited" (L) means that limited information is provided for the particular group of processes.

According to this table, there is no methodology with a full conformance to our criteria except from *Consensus-based* and GM methodologies. However the *Consensus-based* ontology approach to ontology engineering may require more time and effort to deployment as opposed to other approaches due to the iterative cycles of the consensus building mechanism but this tradeoff is expected to be improved in the long term. In addition, the GM methodology emphasizes the notion of collaboration in the development process, particularly during knowledge acquisition. The GM knowledge acquisition relies heavily on interaction; the higher the level of interaction amongst domain experts, the more refined the specific models are likely to be.

The purpose of the benchmarking process we conducted was to figure out how sufficiently each methodology conforms to the proposed evaluation criteria. The process adopted for this purpose was similar to the one adopted in the *OntoWeb* project. A description of methodologies for developing, maintaining, evaluating and re-engineering ontologies is provided in the public deliverable D1.4 of the *OntoWeb* project [3], while a thorough survey can be also found in [23]. Our evaluation procedure presented in this paper extended the one used in *OntoWeb* by adding nine more methodologies to the existing ones.

5 Conclusions and Future Work

In this paper we presented a comparative analysis whose goal is to benchmark a set of different ontology management and development methodologies according to a set of criteria. For the deployment of our evaluation framework we have used the one adopted by the *Ontoweb* project as a basis. Because there are quite a few survey papers on

Table 1. Comparison of various ontology development methodologies with respect to the new criteria based on the IEEE 1074-1995 standard processes. The following abbreviations are used. NP: not proposed, D: described, L: limited, ND: not described, P: proposed.

	Processes distinguished in the IEEE 1074-1995 standard and corresponding criteria				
Methodologies	**Project management**	**Pre-development**	**Development**	**Post-development**	**Integral**
Cyc	NP	NP	NP	NP	L
Uschold & King's	ND	ND	L	NP	L
TOVE	ND	ND	D	NP	L
METH-ONTOLOGY	D	D	D	P-ND	D
KACTUS	D	D	D	NP	L
SENSUS	D	D	D	NP	D
CommonKADS	L	D	D	NP	L
OTK	L	D	L	NP	L
ROD	D	D	D	D	L
Holsapple	D	D	D	NP	L
DOGMA	D	D	D	NP	D
DILIGENT	NP	D	NP	L	D
HCOME	D	D	D	P-ND	D
UPON	NP	D	D	NP	L
Consensus-based	D	D	D	D	D
GM	D	D	D	D	D
iCapturer	D	P-ND	D	D	D
NeOn	NP	L	NP	D	D

methodology, we provided a brief description of the most well known approaches for building ontologies both from scratch, or reusing ontologies from existing ontology repositories. Finally we evaluated them according to the set of criteria proposed based on the IEEE 1074-1995 standard.

Moreover, our framework extends the *Ontoweb* framework by also including nine additional methodologies. The evaluation results reveal conformance of the evaluated methodologies according to the different criteria proposed in this paper. We have seen that Consensus-based and GM methodologies comply almost perfectly with our criteria. However, *Consensus-based* and GM are difficult to be applied. *Consensus-based* ontology approach requires more time and effort to deployment due to the iterative cycles of the consensus building mechanism. In addition, the GM methodology emphasizes the notion of collaboration in the development process, particularly during knowledge acquisition. The fact that GM knowledge acquisition relies heavily on interaction makes this methodology difficult to use and understand, especially for users who want to apply/reuse it (for example, for their own application). Therefore, a solution that balances between performance, usability and conformance to criteria should be mainly sought among *SENSUS*, *METHONTOLOGY*, *DILLIGENT* and *HCOME*.

The comparative analysis proposed in this paper extended the Ontoweb framework that has been used as evaluation framework for ontology development methodologies by

adding further criteria that drive the comparison. These criteria support domain experts, users, knowledge engineers and ontology engineers in collaboratively restructuring a shared ontology. Moreover, the presented evaluation framework guides the participants in a perfect way through the ontology development lifecycle, allowing for personalization and taking into account specific criteria as the proposed ones. Our future plans include the establishment of quantitative metrics to measure the conformance of the benchmarked methodologies with respect to the different evaluation criteria.

A real case of how the aforementioned criteria can be applied on a real ontology is going to be presented. The described process will be applied to a set of existing ontologies. The expected outcome of the presented case study and evaluation will form the basis for the development of a concrete methodological approach and a set of criteria for ontology evaluation.

References

1. IEEE Standard for Developing Software Life Cycle Processes. IEEE Computer Society, New York (USA). April 26, 1996.
2. K.K. Breitman, M.A. Casanova and W. Truszkowski, "Methods for Ontology Development". Semantic Web: Concepts, Technologies and Applications, Part III, Chapter 8, pp. 155-173, Springer, 2007.
3. OntoWeb Project IST-2000-29243, Deliverable 1.4: "A Survey on Methodologies for Developing, Maintain-ing, Evaluating and Reengineering Ontologies", Available online: http://www.ontoweb.org/About/Deliverables/OverviewProjectPhase3/D1.4-v1.0.pdf
4. M. Delanda, Intensive science and virtual philosophy, Continuum, New York, 2002.
5. D.B. Lenat and R.V. Guha, "Building large knowledge-based systems". Addison-Wesley Publising Com-pany, Inc. 1990.
6. A.L. Rector, "Modularisation of domain ontologies implemented in description logics and related formal-isms including OWL". In Proceedings of the 2nd international Conference on Knowledge Capture (Sanibel Island, FL, USA, October 23 - 25, 2003).
7. M. Uschold and M. King, "Towards a Methodology for Building Ontologies". In Proc. of Workshop on Basic Ontological Issues in Knowledge Sharing (held in conjunction with IJ-CAI'95), 1995, Montreal, Can-ada.
8. M. Gruninger and M. Fox, "Methodology for the Design and Evaluation of Ontologies". In Proc. of Work-shop on Basic Ontological Issues in Knowledge Sharing (held in conjunction with IJCAI'95), 1995, Mont-real, Canada.
9. M. Fernández-López, A. Gómez-Pérez and N. Juristo, "METHONTOLOGY: From Ontological Art Towards Ontological Engineering". In Proc. of Spring Symposium on Ontological Engineering (AAAI'97), pp. 33-40, 1997, Stanford, CA, USA.
10. B. Swartout P. Ramesh, K. Knight and T. Russ, "Toward Distributed Use of Large-Scale Ontologies". In Proc. of Spring Symposium on Ontological Engineering of AAAI, pp. 138-148, March 24-26, 1997, Stan-ford, CA, USA.
11. G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. Van de Velde and B. Wielinga, "Knowledge Engineering and Management - The CommonKADS Methodology". MIT Press. (1999)
12. S. Staab, H.P. Schnurr, R. Studer and Y. Sure, "Knowledge processes and ontologies". IEEE Intelligent Systems 16 (1) (2001) 26-34.
13. L. Zhou, Q. E. Booker and D. Zhang, "Toward Rapid Ontology Development for Underdeveloped Domains". HICSS 2002: 106

14. C.W. Holsapple and K.D. Joshi, ”A collaborative approach to ontology design”. Commun. ACM 45 (2002) 42-47
15. P. Spyns, R. Meersman and M. Jarrar, ”Data modelling versus ontology engineering”. SIGMOD Record Special Issue 31(4), 12-17. (2002)
16. H. S. Pinto, S. Staab and C. Tempich, ”DILIGENT: Towards a fine-grained methodology for Distributed, Loosely-controlled and evolving Engineering of oNTologies”. ECAI 2004: 393-397.
17. K. Kotis and G. Vouros, ”Human-Centered Ontology Engineering: the HCOME Methodology”. International Journal of Knowledge and Information Systems (KAIS), 10(1): 109-131 (Published Online First: 9 Sept. 2005)
18. M. Missikoff and R. Navigli, ”Applying the unified process to large-scale ontology building”. In Proceed-ings of 16th IFAC World Congress (IFAC) (pp. 61-96). Amsterdam: Elsevier. (2005)
19. S. Karapiperis and D. Apostolou, ”Consensus building in collaborative ontology engineering processes”. Journal of Universal Knowledge Management, 1(3), 199-216 (2006).
20. A. Garcia Castro, P. Rocca-Serra, R. Stevens, C. Taylor, K. Nashar, M. Ragan and S. Sansone, ”The use of concept maps during knowledge elicitation in ontology development processes - the nutrigenomics use case”. BMC Bioinformatics, 7, 267-281. (2006)
21. Good, B., Tranfield, E.M., Tan, P.C., Shehata, M., Singhera, G., Gosselink, J., Okon, E.B., Wilkinson, M., ”Fast, cheap, and out of control: A zero curation model for ontology development”. In: Pacific Symposium on Biocomputing. (2006)
22. M. C. Su?rez-Figueroa, K. Dellschaft, E. Montiel-Ponsoda, B. Villaz?n-Terrazas, Z. Yufei, G. Aguado de Cea, A. Garc?a, M. Fern?ndez-L?pez, A. G?mez-P?rez, M. Espinoza, M. Sabou. NeOn Deliverable D5.4.1: ”NeOn Methodology for Building Contextualized Ontology Networks”. NeOn Project. Available online: http://www.neon-project.org. February 2008.
23. C. Tempich, H. S. Pinto and S. Staab, ”Ontology Engineering Revisited: An Iterative Case Study”. ESWC 2006: 110-124.

A pan-European repository: SEMIC.EU as the point of reference for eGovernment ontologies

Klaus Reichling, Martin Luts, Renke Fahl-Spiewack

Semantic Interoperability Centre Europe,
]init[AG für digitale Kommunikation, Berlin, Germany
ELIKO Competence Centre in Electronics-, Info- and Communication Technologies, Tallinn, Estonia; Department of Informatics, Tallinn University of Technology, Estonia

Abstract. The Semantic Interoperability Centre Europe (SEMIC.EU) is a Service of the European Commission to foster seamless data exchange. One of the principal objectives of the initial project setup was the creation of a repository of semantic interoperability assets for pan-European eGovernment projects. eGovernment arrived at the opportunities of semantic technologies. A recent poll by SEMIC.EU showed: Over 80 per cent of respondents indicated they have used or are going to use ontologies in the next 12 months. European public administration has realized the potential of ontological solutions for its purposes.

Keywords: semantic interoperability, European Commission, eGovernment, repository, ontologies, ISA, IDABC

1 Introduction

The Semantic Interoperability Centre Europe, SEMIC.EU, is a service offered by the European Commission: It facilitates the reuse of syntactic and semantic assets needed for semantic interoperability. It was initiated in the framework of the programme 'Interoperable Delivery of European eGovernment Services to public Administrations, Businesses and Citizens' (IDABC) and is carried on under the successor programme "Interoperability Solutions for European Public Administrations", ISA[1].

A standardised clearing process, supported by platform functionalities, governs the evolution of the data models, XML schemas, code lists, taxonomies and ontologies which are shared through the open repository.

[1] http://ec.europa.eu/isa

The service addresses seamless data exchange in Europe, particularly in public administration. It is understood that countries have different administrative, technical and linguistic backgrounds. Solutions for the technical, semantic, and organisational dimensions of interoperability are needed. SEMIC.EU's focus is on the vital semantic dimension of interoperability.

As a collaborative service and with a repository of assets for semantic interoperability, SEMIC.EU is a catalyst for the preservation of meaning in the data exchanged at all levels of administration.

2 The asset principle

The SEMIC.EU web portal comprises an open repository of "interoperability assets". The term "interoperability asset" describes a resource that supports the exchange of data in distributed information systems, roughly to be understood as data models that help overcome differences in the systems involved in the exchange of certain data.

SEMIC.EU's most common semantic specifications used in interoperability assets are code lists, taxonomies and also ontologies including their respective mappings.

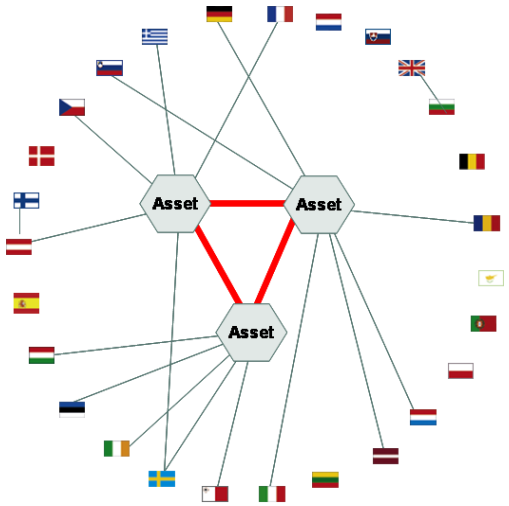


Fig. 1 Multilateral assets and mappings

They are collected, quality checked and made available for download and reuse in the web repository. The assets are taken through an open clearing process [SEM01].

The common features of all these interoperability assets are their design and suitability for concrete data exchange and data modeling purposes in a wider eGovernment context.

2.1 Life cycle of an interoperability asset

SEMIC.EU has defined a collaborative process of incremental quality enhancement and adaptation to a comprehensive set of criteria. Any asset submitted to the repository enters this *clearing process*.

An asset is a package of related documents that form a specification. The contained code list, taxonomy, XML schema or ontology is just a component of the asset as a package.

Fig. 2. Screenshot: Asset Repository

From a rough draft to a certified recommended means of facilitating data interchange, the process requires actions from the owner of the asset, SEMIC.EU as the broker and quality assurer, the general public for consultation and enhancement and expert communities for validation and approval. Technical, semantic and pragmatic aspects are assessed during all steps. Assets of interoperability and their respective releases are scrutinised by a staff member - the clearing process manager, as well as through public consultation and by experts in the domain of the asset.

During the process, assets are labelled with their current state of quality assurance. The successive stages of the process depend on actions by the asset agent (the person in charge of the asset) and the clearing process manager, supported by the platfrom (see chapter 3).

Although every asset has its own, individual life cycle, the paths and development stages are defined by the clearing process of the SEMIC.EU platform [SEM02].

Assets pass a number of successive stages consisting of quality assessment and feedback. A certain label indicates the maturity level of an asset (or one of its releases) – known as the "asset/release state":

Unpublished

For an unpublished asset to be submitted to SEMIC.EU, there are no technical or formal requirements other than the registration of the asset agent.

The asset agent must submit a set of metadata and select a topical category (domain) for an asset, but has not requested its publication yet.

The asset is not displayed in any search results or lists outside the asset agent's personal area on the website. It can only be viewed by the agent and the Clearing Process Manager.

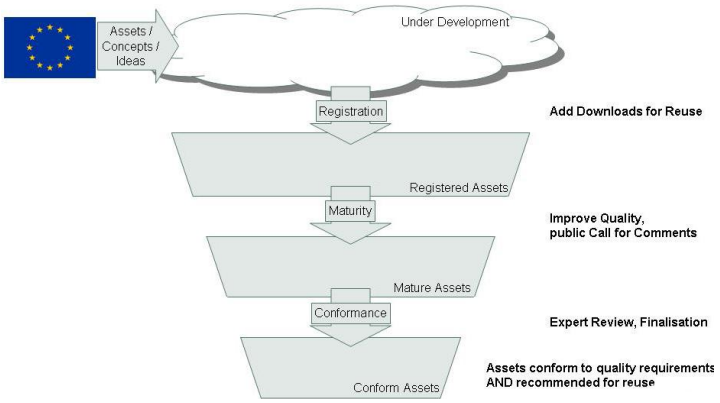


Fig. 3. Basic three-step scheme of the clearing process

Development

There are still no technical requirements, but the asset is now published and visible to all users.

To put an asset "under development", the asset agent needs to request its publication which will then be approved or rejected by the clearing process manager. This is done via repository functionalities.

This state is chosen by the agent if he or she wants to get feedback and support by other users. With an asset "under development", asset agents can make use of community features and seek advice from others in the development of their data model. For each published asset, a dedicated forum thread is created at www.semic.eu.

Registered

This is the first state of the actual clearing process since for this state to be reached, an asset needs substance (i.e. a first release):

The asset agent must upload the actual artefacts (data model files, a licence specification and other documentation) or provide a link where this information can be obtained.

If the clearing process manager approves the publication of the release, both the release and the asset are marked as "registered".

Mature

The requirements of negotiation and collaboration take centre stage here: An asset marked as "mature" has taken the important first step of the quality assurance and enhancement process.

At least one release of an asset has been approved as mature by the clearing process manager and after a community review. The artefacts it consists of meet quality criteria concerning their completeness, syntactic requirements and usability for practical applications.

The focus in granting "mature" state to an asset is on its potential for reuse.

This is a cyclic process in which the asset enhances its quality incrementally (in its respective releases).

Conform

An asset accepted as conform by the clearing process manager and the conformance committee – a group of domain experts - has reached the highest possible quality state, i.e. one of its releases was accepted as conform.

From a clearing process perspective, the development of an asset ends here. It is considered conform to all the criteria defined in the Clearing Process Framework.

Modifications (especially for reuse in other contexts) will make it a new release or, in most cases, a new asset which will undergo a separate clearing process.

These stages, the community review and the role of the conformance committee reflect the interactive nature of the entire process as well as the room for negotiation about appropriate semantic properties, granularity, usability and other quality criteria foreseen in the process. The basic process is illustrated in Fig. 3.

The controlled vocabularies [SEM03] and data models provided enter the clearing process, i.e. they are subject to a quality assurance procedure and actual peer review, possibly resulting in the publication of revised and enhanced versions. Through this instrument, SEMIC.EU introduces a new layer of negotiation in the creation of semantic interoperability.

3 Roles and platform features in the clearing process

The clearing process involves people in various roles that are given to registered users. They are supported by the SEMIC.EU platform in executing their tasks. With respect to their duties and rights in the context of an interoperability asset, roles are assigned to these users. The roles are attached to permissions which enable the users to execute tasks on the SEMIC.EU platform. Slightly simplified, the roles are divided into: audience, asset provider (agent) and SEMIC.EU as an organisation (including the clearing process manager). These groups carry out the clearing process in its successive stages.

Assistance is provided by SEMIC.EU in questions of documentation, licensing and publication of assets. After the publication of an asset, the owner can request additional quality assessment.

Asset assessment

Asset agents receive feedback on their assets from the SEMIC.EU clearing service, practical advice on development and avoidance of potential semantic conflicts compiled in the form of a comprehensive SEMIC.EU asset assessment report.

Asset management / Repository

The platform manages the versioning and assists in the collaborative development of the asset. When the asset agent uploads an asset to the repository, the asset receives an initial version number, incremented each time a new version of the asset is uploaded or the asset progresses to another state in the maturity process. Using the forum, interested users can discuss the design and development decisions.

Asset refinement

During the asset refinement the clearing manager gives feedback on the asset concerning its design, documentation, modularisation and semantic accuracy. A focus in the refinement of assets is their respective potential for reuse.

Asset harmonisation

During the asset harmonisation assets with similar focus are studied and if possible consolidated. The SEMIC.EU clearing manager works together with the asset-agents and requests feedback from partners, experts and interested stakeholders.

Asset development assistant

Additional support in the development process can be requested via the asset development assistant, an interactive feature linked to the registration and development processes. The assistant suggests issues to consider, gives examples and guides the user through all relevant phases of asset development, asset clearing, licensing and quality improvement. Common user requests and issues of general interest are successively included in the assistant functionality.

Contents on the platform are interlinked and related (e.g. actors and assets), so that users can recognise the relationships of artefacts (derivates, similar content or use case). One of the most important cases is the reference from a mapping asset to its respective source and target artefacts (taxonomy or ontology).

4 Exploiting the potential of ontologies

The potential of ontologies to generate semantic interoperability in eGovernment services and general data exchange is commonplace. However, empirical data on the actual use of ontologies and their acceptance by practitioners has yet to be gathered.

The activities coordinated by SEMIC.EU show a more widespread use of single data models, code lists and taxonomies in every-day eGovernment contexts. They are shared through the open repository of SEMIC.EU, undergo an open, peer-review

based quality assurance process and are the object of community work and coordination.

However, a recent poll by SEMIC.EU among its user base has shown that well over 80 per cent of the respondents claimed to have used or are planning to use ontologies for data exchange within the next 12 months[2].

Based on this we expect that more than 30 eGovernment projects will request for SEMIC.EU coaching services to develop, implement and reuse of ontologies in practical eGovernment projects within the next year.

In a Union of 27 Member states with a number of other associations, candidate countries, the European Economic Area, etc., federating existing sources and making them accessible through a central resource, is a pivotal task.

As a European centre for eGovernment services, it is SEMIC.EU's duty and aim to establish the link between ontologies and eGovernment practice.

From the point of view of government users the benefit is often not apprehensible, in contrast to other semantic specifications with simpler structures like code lists and glossaries.

Public administration bodies themselves have not shown readiness for implementation or usage of ontologies (or related methods).

5 eGovernment requires reliability, soundness, usability

All SEMIC.EU's activities and initiatives are performed in the public interest. Therefore they are always driven by the demand for practical implementation. This has a number of consequences also for strategic and organizational considerations:

SEMIC.EU's mission is not to be fast or avant-garde but rather solid, stable and reliable.

The demands of public services require that solutions are concrete, reliable, in line with common standards and legislation, and – to a certain degree – transferable. The latter is a specific European requirement due to the heterogeneous linguistic and legislative landscape in the European Union despite the existence of European directives and regulations[3].

What about SEMIC.EU's relationship to the semantic web community and existing semantic technologies? Knowledge is indeed extracted from silos, made available and accessible but only as a function of its usability and purposefulness in matters of public administration and data exchange for eGovernment services. Peer-reviews and community work warrant accuracy and practicability of the proposed solutions.

---

[2] The survey was conducted via e-mail among 900 users (experts in eGovernment, IT) of SEMIC.EU representing predominantly EU Member States. The response rate was 20 percent. Respondents were asked to choose one out of three statements ("Yes, we have already used ontologies", "Yes, we plan to implement ontology based applications within the next 12 months", "No, ontologies have never been relevant in the context of our work").

[3] Ferrario and Guarino givan an assessment of the potential of ontologies in services contexts [6]

5.1 Is government ready to work with ontologies?

Against this background, we can state that eGovernment has caught up – it has, in many cases and ways, embraced the opportunities of semantic technologies. The poll mentioned above illustrates the demand and willingness of an eGovernment audience to follow suit, or, respectively, lead the way.

European public administration has realised the potential of ontological solutions for its purposes and for the extension of its services and efficiency measures.

Austria has once more proven to be a frontrunner in the adoption of technologies for government services. The Austrian chancellery and municipalities already offer ontology-based citizen services [7].

The European Commission's DG Environment is developing a novel information platform and service, making use of dispersed information from various domains and resources in order to generate added value and integrated information in climate change matters.

5.2 Are ontologies ready to deliver to government?

Researchers and developers are now called on, it is their task to produce usable ontologies that meet the needs and requirements of practitioners, i.e. secure, reliable, proven ontologies and methodologies that have the power to be used in real-life scenarios. Theory is a precondition, practicability is what counts at the end of the day.

Moreover, it will be advantageous to create and find links between ontologies and other types of semantic interoperability assets already available on SEMIC.EU.

For an ontology to be of use to administrative services, quality requirements must be clear, precise and comprehensive.

6 Making implementation possible: Federation and Reuse

The aspects of reuse and exchange are imperative for SEMIC.EU's approach. The Semantic Interoperability Centre Europe is therefore not only organized around the online platform, but also with its communities and as a coaching and clearing service.

How can eGovernment projects and ontology developers benefit from each other? SEMIC.EU provides the organizational structures for coordination and federation of existing resources [SEM04] & [EPR01]. The approach of collaborative issue solving and network building creates an environment that can work as an interface between theory and practical requirements.

At the same time it is an information space where latest developments, unresolved issues and new ideas can be shared.

Much as it is SEMIC.EU's task to coordinate cross-country and pan-European cooperation, it is also vital to monitor and display trends and developments. This is, however, always with a view to quality and practicability criteria.

What does SEMIC.EU have to offer to the scientific community and to practitioners at the same time? It is the European source for cooperation partners, experiences, real life application of technologies (proof of concept), bringing together dispersed initiatives and knowledge. Making solutions accessible as a clearinghouse is one part of this function.

In order to turn this general mission into actual benefit, ontologies (including their drafts and beta versions) must be documented and available at SEMIC.EU in order to become ready for use (see Fig. 4 for the total number of assets available through SEMIC.EU).

Only if this is accepted and put into practice, can the SEMIC.EU community yield results from its opportunities for joint and coordinated development of ontologies, to double-check against practice requirements and feasibility.

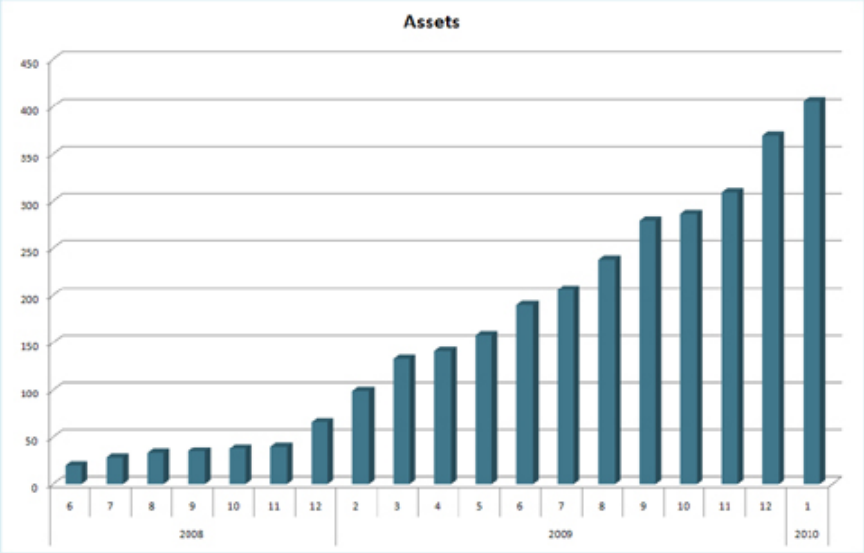


Fig. 4 Total assets registered at SEMIC.EU

6.1 What does SEMIC.EU not do?

SEMIC.EU does not develop ontologies itself. Coordination, coaching, investigation, demand and applicability are the main driving forces for ontology coordination through SEMIC.EU. Therefore, it offers academics, government officials, IT professionals and project managers assistance in

- using an existing and established infrastructure with access to decision-makers, practitioners, voices in both technical and administrative/political issues.

- Building on a network and getting the backup of the central European administrative body, the European Commission.

7 SEMIC.EU – The role as a point of collaboration for pan-European eGovernment initiatives

SEMIC.EU aims to facilitate seamless data exchange in Europe, particularly in public administration. Therefore questions of semantic interoperability have to be solved. Ontologies are one of the most relevant concepts to enable semantic interoperability.

Currently SEMIC.EU initiates collaboration with other Member States initiatives and domain specific repositories to facilitate the integration and federation of different repositories. Cooperations have been established with ontology repositories such as the United Nation's Food and Agricultural Organization's AIMS (Agricultural Information Management Standards) and the United States National Center for Biomedical Ontology (NCBO).

Looking from the practical point there are quite a lot of open questions to be solved.

- What is the most promising and useful strategy to link them?
- Which metadata are relevant and need to be shared?
- How can we establish interoperability between ontology repositories?

Our next step is to create a common metadata schema that allows for data exchange between different repositories. Furthermore it will be necessary to develop different scenarios for repository integration, and define use-cases and collect requirements for these scenarios. In the context of ontology development and mapping, this will also mean that visualisation, relationship definition and navigation support can be realised.

SEMIC.EU invites all other initiatives of open repository development to collaborate.

References

1. [SEM01] Fraunhofer ISST and]init[.Quality Framework for Interoperability Assets. European Commission – IDABC. 2008.
2. [SEM02] Fraunhofer ISST and]init[. Vision of the Clearing Process. European Commission – IDABC. 2008.
3. [SEM03] Fraunhofer ISST and]init[. Study on Methodology. European Commission – IDABC. 2009.
4. [SEM04] Community on SEMIC.EU Technology Reuse http://www.semic.eu/semic/view/snav/network/Communities/Technology-reuse.xhtml?cid=378223
5. [EPR01] SEMIC.EU Technology re-use Community http://www.epractice.eu/community/semictechreuse
6. Ferrario, R.; Guarino, N.: Towards an Ontological Foundation for Services Science. In: J. Domingue, D. Fensel & P. Traverso (eds.) Future Internet - FIS 2008, First Future Internet Symposium, Vienna, Austria, September 29-30, 2008, Revised Selected Papers (Lecture Notes in Computer Science 5468), pp. 152-169 (2008)
7. Salhofer, P., Stadlhofer, B.; Tretter, G.: Ontology Driven e-Government. In: P. Hahamis (ed.), Proceedings of the 9th European Conference on e-Government, London, United Kingdom, June 29-30, 2009, Reading: API, pp. 540-549 (2009)