D1.2.2.1.1 Benchmarking the interoperability of ontology development tools using RDF(S) as interchange language

Raúl García-Castro (UPM)

with contributions from:
York Sure (UKARL)
Markus Zondler (UKARL)
Olivier Corby (INRIA)
Jesús Prieto-González (UPM)
Elena Paslaru Bontas (FU Berlin)
Lyndon Nixon (FU Berlin)
Malgorzata Mochol (FU Berlin)

Abstract.
EU-IST Network of Excellence (NoE) IST-2004-507482 KWEB
Deliverable D1.2.2.1.1 (WP 1.2 & WP2.1)

This deliverable describes the benchmarking of the interoperability of ontology development tools using RDF(S) as interchange language that has taken place in Knowledge Web, including the analysis of the results obtained.
Keyword list: benchmarking, benchmark suite, interoperability, RDF(S)

Document Identifier	KWEB/2006/D1.2.2.1.1/v1.5
Project	KWEB EU-IST-2004-507482
Version	v1.5
Date	3. August, 2006
State	final
Distribution	public

Knowledge Web Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2004-507482.

University of Innsbruck (UIBK) - Coordinator
Institute of Computer Science
Technikerstrasse 13
A-6020 Innsbruck
Austria
Contact person: Dieter Fensel
E-mail address: dieter.fensel@uibk.ac.at

École Polytechnique Fédérale de Lausanne (EPFL)
Computer Science Department
Swiss Federal Institute of Technology
IN (Ecublens), CH-1015 Lausanne
Switzerland
Contact person: Boi Faltings
E-mail address: boi.faltings@epfl.ch

France Telecom (FT)
4 Rue du Clos Courtel
35512 Cesson Sévigné
France. PO Box 91226
Contact person : Alain Leger
E-mail address: alain.leger@rd.francetelecom.com

Freie Universität Berlin (FU Berlin)
Takustrasse 9
14195 Berlin
Germany
Contact person: Robert Tolksdorf
E-mail address: tolk@inf.fu-berlin.de

Free University of Bozen-Bolzano (FUB)
Piazza Domenicani 3
39100 Bolzano
Italy
Contact person: Enrico Franconi
E-mail address: franconi@inf.unibz.it

Institut National de Recherche en Informatique et en Automatique (INRIA)
ZIRST - 655 avenue de l'Europe -
Montbonnot Saint Martin
38334 Saint-Ismier
France
Contact person: Jérôme Euzenat
E-mail address: Jerome.Euzenat@inrialpes.fr

Centre for Research and Technology Hellas / Informatics and Telematics Institute (ITI-CERTH)
1st km Thermi - Panorama road
57001 Thermi-Thessaloniki
Greece. Po Box 361
Contact person: Michael G. Strintzis
E-mail address: strintzi@iti.gr

Learning Lab Lower Saxony (L3S)
Expo Plaza 1
30539 Hannover
Germany
Contact person: Wolfgang Nejdl
E-mail address: nejdl@learninglab.de

National University of Ireland Galway (NUIG)
National University of Ireland
Science and Technology Building
University Road
Galway
Ireland
Contact person: Christoph Bussler
E-mail address: chris.bussler@deri.ie

The Open University (OU)
Knowledge Media Institute
The Open University
Milton Keynes, MK7 6AA
United Kingdom
Contact person: Enrico Motta
E-mail address: e.motta@open.ac.uk

Universidad Politécnica de Madrid (UPM)
Campus de Montegancedo sn
28660 Boadilla del Monte
Spain
Contact person: Asunción Gómez Pérez
E-mail address: asun@fi.upm.es

University of Karlsruhe (UKARL)
Institut für Angewandte Informatik und Formale
Beschreibungsverfahren - AIFB
Universität Karlsruhe
D-76128 Karlsruhe
Germany
Contact person: Rudi Studer
E-mail address: studer@aifb.uni-karlsruhe.de

University of Liverpool (UniLiv)
Chadwick Building, Peach Street
L697ZF Liverpool
United Kingdom
Contact person: Michael Wooldridge
E-mail address: M.J.Wooldridge@csc.liv.ac.uk

University of Manchester (UoM)
Room 2.32. Kilburn Building, Department of Computer
Science, University of Manchester, Oxford Road
Manchester, M13 9PL
United Kingdom
Contact person: Carole Goble
E-mail address: carole@cs.man.ac.uk

University of Sheffield (USFD)
Regent Court, 211 Portobello street
S14DP Sheffield
United Kingdom
Contact person: Hamish Cunningham
E-mail address: hamish@dcs.shef.ac.uk

University of Trento (UniTn)
Via Sommarive 14
38050 Trento
Italy
Contact person: Fausto Giunchiglia
E-mail address: fausto@dit.unitn.it

Vrije Universiteit Amsterdam (VUA)
De Boelelaan 1081a
1081HV. Amsterdam
The Netherlands
Contact person: Frank van Harmelen
E-mail address: Frank.van.Harmelen@cs.vu.nl

Vrije Universiteit Brussel (VUB)
Pleinlaan 2, Building G10
1050 Brussels
Belgium
Contact person: Robert Meersman
E-mail address: robert.meersman@vub.ac.be

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to writing parts of this document:

Centre for Research and Technology Hellas
École Polytechnique Fédérale de Lausanne
Free University of Bozen-Bolzano
Institut National de Recherche en Informatique et en Automatique
Learning Lab Lower Saxony
National University of Ireland Galway
Universidad Politécnica de Madrid
University of Karlsruhe
University of Manchester
University of Sheffield
University of Trento
Vrije Universiteit Amsterdam
Vrije Universiteit Brussel

Changes

Version	Date	Author	Changes
0.1	11.04.06	Raúl García-Castro	Created
0.2	03.05.06	Raúl García-Castro	Inserted the import and export results for Protégé and WebODE
0.3	18.05.06	Raúl García-Castro	Inserted the interoperability results for Protégé and WebODE
0.4	22.05.06	Raúl García-Castro	Inserted the instantiation of the methodology and the benchmark suites
0.5	23.06.06	Olivier Corby, Raúl García-Castro, Jesús Prieto, York Sure, and Markus Zondler	Inserted the Corese, Jena and KAON results
0.6	27.06.06	Raúl García-Castro	Inserted the analysis of the results
0.7	29.06.06	Raúl García-Castro	Inserted the recommendations and conclusions
0.8	30.06.06	Elena Paslaru Bontas, Lyndon Nixon and Malgorzata Mochol	Inserted the RDF(S) interoperability requirements in the use cases
1.0	30.06.06	Raúl García-Castro	First version of the document sent to the Quality Assessor (Holger Wache)
1.1	06.07.06	Raúl García-Castro	Included the comments from Rosario Plaza
1.2	07.07.06	Raúl García-Castro	Included the comments from the Quality Assessor (Holger Wache)
1.3	17.07.06	Raúl García-Castro and Jesús Prieto	Inserted the Sesame results
1.4	26.07.06	Raúl García-Castro	Included the comments from the Quality Controller (Sean Bechhofer)
1.5	03.08.06	Raúl García-Castro	Included the comments from the Quality Assurance Coordinator (Francisco Martin-Recuerda)

Executive Summary

In 2005, a new activity for benchmarking the interoperability of ontology development tools using RDF(S) as interchange language was started in Knowledge Web; its goal was to learn about the actual interoperability between these tools and, if possible, to improve it.

This deliverable includes the work performed in workpackages 1.2 and 2.1 during this benchmarking activity and presents an overview of the benchmarking and its main results; it comprises the following topics:

- Instantiation of the Knowledge Web benchmarking methodology for carrying out the benchmarking.
- Definition of the benchmark suites used in the benchmarking.
- Analysis of the results obtained in the benchmarking.
- Recommendations for users and developers based on this analysis.
- Analysis of the feasibility of RDF(S) interoperability in the business use cases.
- Detailed results of the benchmarking.

Contents

Chapter 1

Introduction

by RAÚL GARCÍA-CASTRO

In Knowledge Web, different benchmarking activities are being (and will be) performed to improve ontology tools and to offer recommendations on these tools for both research and industry users. One of these benchmarking activities is the interoperability of ontology development tools using RDF(S) as interchange language which started in 2005, and to which the content of this deliverable is related.

This deliverable originated from the joint work performed by WP 1.2 in the industry area and by WP 2.1 in research. In the latter, the members of WP 2.1 developed the benchmarking methodology for ontology tools, which we have followed in this benchmarking activity [García-Castro *et al.*, 2004], and the benchmark suites used in the experimentation [García-Castro, 2005], whereas the members of WP 1.2 have organised the benchmarking activity, performed the experimentation over the tools, and analysed the results.

By the time of writing this deliverable, six tools are participating in the benchmarking three of which are ontology development tools: KAON, Protégé using its RDF backend, and WebODE, and three are RDF repositories: Corese, Jena and Sesame. Another ontology development tool, OntoStudio, is also taking part but it is not considered in this deliverable because its complete evaluation results are not yet available.

The benchmarking methodology proposes to produce two documents in the benchmarking activity: the *Experiment Report* which presents the analysis of the results of the experiments; and the *Benchmarking Report* which gives an understandable summary of the benchmarking activity and its results and conclusions. These two documents are included in this deliverable.

The broad scope of the deliverable results in a large number of pages. To facilitate its reading, we have divided it into two documents: one contains the chapters of the deliverable whereas the other contains the appendixes with the experimentation results in detail.

The deliverable is composed of different chapters oriented to different audiences.

Therefore, readers can just read the chapters of their interest following these guidelines:

- Chapter 2 is an extension of [García-Castro and Gómez-Pérez, 2006a] and presents how the RDF(S) interoperability benchmarking was conducted following the Knowledge Web benchmarking methodology. It is intended to provide a summary of the benchmarking activity to organisation managers, benchmarking teams, tool developers, and tool users both from academia and industry.
- Chapter 3 is an extension of [García-Castro and Gómez-Pérez, 2005a] and states the method followed to define the three benchmark suites used in the benchmarking. It can provide inspiration to tool developers for developing new benchmark suites or it can be taken as a guide for executing the benchmarking experiments.
- Chapters 5, 4 and 6 summarize the results of executing the export, import and interoperability benchmark suites respectively on the tools participating in the benchmarking. They present a brief overview of the results to tool developers and users. A detailed description of the results is included in the appendixes. The results of the Protégé ontology editor will be presented in [García-Castro and Gómez-Pérez, 2006b].
- Chapter 7 analyses the interoperability needs of the Knowledge Web industry use cases.
- Chapter 8 provides recommendations for users on the interoperability of ontology development tools, for developers on how to implement interoperability on the tools, for industry on how to accomplish the Knowledge Web use cases, and for anybody interested in this issue on how to organize a benchmarking activity.
- The appendixes, as mentioned above, offer detailed results of the execution of the experiments; they can be consulted by users or developers whenever they have a specific interoperability problem in their tools. They are located in a separate document that can be downloaded from the Knowledge Web portal[1].

[1]http://knowledgeweb.semanticweb.org/

Chapter 2

Benchmarking RDF(S) interoperability

by RAÚL GARCÍA-CASTRO

This chapter presents how the RDF(S) interoperability benchmarking was organized and carried out following the methodology for benchmarking ontology tools developed in Knowledge Web [García-Castro *et al.*, 2004].

Figure 2.1 shows the three phases that compose the benchmarking methodology and the tasks to be performed in each phase. As we have already mentioned, this document comprises both the experiment and the benchmarking reports. Therefore, this chapter includes the instantiation of this methodology from the beginning of the benchmarking activity to the end of the *Experiment* phase, which is the last task performed before writing this deliverable.

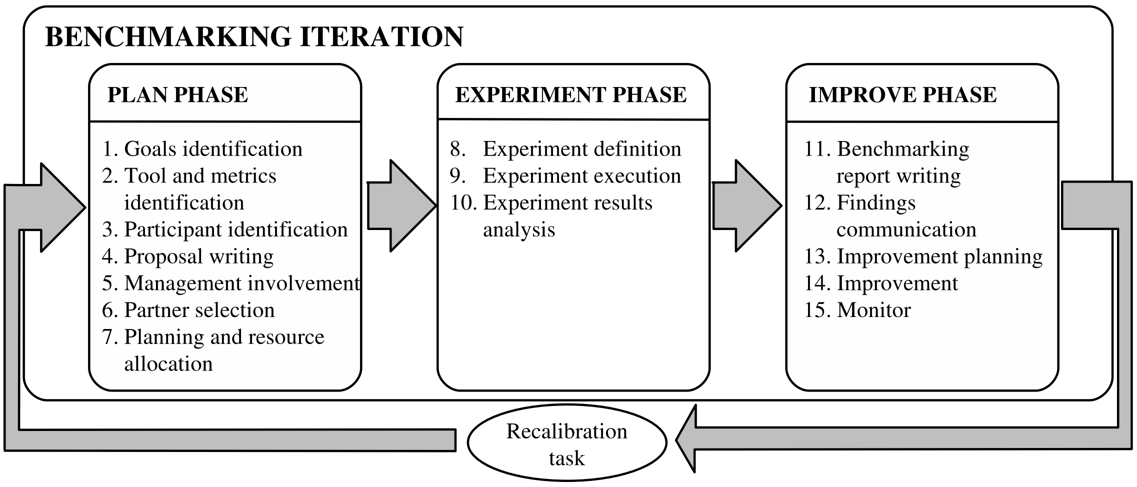


Figure 2.1: The benchmarking methodology for ontology tools

2.1 Plan phase

2.1.1 Goals identification

Raúl García Castro, from the UPM, took the role of the benchmarking initiator and prepared the benchmarking, carrying out the first tasks of its process.

The goals for benchmarking the interoperability of ontology development tools are related to the benefits pursued through it, and these are:

- To evaluate and improve the interoperability of these tools.
- To produce recommendations on the interoperability of these tools for users.
- To acquire a deep understanding of the practices used to develop the importers and exporters of these tools.
- To extract from these practices those that can be considered best practices when developing importers and exporters.
- To create consensual processes for evaluating the interoperability of these tools.

These benefits involve different communities that are related to the ontology development tools, namely, the research community, the industrial community, and the tool developers.

Most of the costs of the benchmarking go to the human resources needed for organising the benchmarking activity and for performing the experimentation on the tools. Other minor costs go to travelling and computers, but they are negligible compared to the aforementioned.

2.1.2 Tool and metrics identification

WebODE [Arpírez *et al.*, 2003] is the ontology engineering platform developed by the Ontology Engineering Group of the UPM and the tool chosen to participate in the benchmarking.

Of the different evaluation criteria to consider when evaluating ontology development tools, i.e., performance, scalability, interoperability, robustness, etc.; we have contemplated only interoperability. An approach for benchmarking the performance and scalability of these tools can be found in [García-Castro and Gómez-Pérez, 2005b].

Achieving interoperability between ontology development tools is not straightforward when these tools do not share a common knowledge model, so their users need to know the effects of interchanging an ontology from one tool to another.

Of the different ways that ontology development tools have to interoperate, either by using an API for accessing ontologies in other tools or by translating the ontologies to an interchange language, we have selected the latter approach as the former is not present in current tools.

In the benchmarking activity that we present in this deliverable, the language used for the interchange is RDF(S) [Brickley and Guha, 2004] and the syntax for serializing the ontologies is the RDF/XML syntax, the syntax for RDF(S) that these tools employ most. A future benchmarking activity inside Knowledge Web will cover the case of using OWL as interchange language [D1.2.2.1.2, 2007].

Interoperability of ontology development tools using an interchange language depends on the capabilities of the tools to import and export ontologies from/to this language. Therefore, the functionalities relevant to the benchmarking are the RDF(S) importers and exporters.

The evaluation criteria must describe in depth the interoperability between the tools, whereas the experiments to be performed in the benchmarking must provide data that inform how the tools comply with these criteria. Therefore, to obtain detailed information about tool interoperability, we need to know:

- The components of the knowledge model of an ontology development tool that can be interchanged with another tool using RDF(S) as interchange language.
- The secondary effects of interchanging these components, such as insertion or loss of information.
- The subset of the components of the knowledge models of ontology development tools that these tools can use to interoperate correctly.

2.1.3 Participant identification

As WebODE is being developed by the Ontology Engineering Group at the UPM, it was quite straightforward to identify and contact the members of the organisation involved in WebODE's RDF(S) importers and exporters and to select among them the members of the benchmarking team.

2.1.4 Proposal writing

The benchmarking proposal, which is being used as a reference along the benchmarking, did not take the form of a paper document but of a web page[1], which is publicly available and includes all the relevant information about the benchmarking: motivation, goals,

[1]http://knowledgeweb.semanticweb.org/benchmarking_interoperability/

benefits and costs, tools and people involved, planning, related events, and a complete description of the experimentation and the benchmark suites.

2.1.5 Management involvement

The benchmarking proposal was presented to the managers of the Ontology Engineering Group and, after their analysis, they agreed on the continuity of the benchmarking and on the allocation of future resources both for performing the experimentation and improving the tool.

2.1.6 Partner selection

Participation in the benchmarking is open to any organisation irrespective of being a Knowledge Web partner or not. To find other best-in-class organisations willing to participate in the benchmarking, the following actions were taken:

- To research different ontology development tools, both freely available and commercial ones, which could export and import to and from RDF(S) and then, to contact the organisations that develop them.
- To announce the interoperability benchmarking and to call for participation through the main mailing lists of the Semantic Web area and through lists specific to ontology development tools.

Table 2.1 presents the ontology development tools capable of importing and exporting RDF(S) found by the time of performing this task. Their developers were directly contacted.

Tool	**Institution**	**URL**
Construct	Network Inference	http://www.networkinference.com/products/construct_it.html
DOE	Inst. National de l'Audiovisuel	http://homepages.cwi.nl/˜troncy/DOE/
InferEd	Intellidimension	http://www.intellidimension.com/pages/site/products/infered/
IsaViz	W3C	http://www.w3.org/2001/11/IsaViz/
KAON	Universitat Karlsruhe	http://kaon.semanticweb.org/
Linkfactory Workbench	Language & Computing	http://www.landcglobal.com/pages/linkfactory.php
OilEd	University of Manchester	http://oiled.man.ac.uk/
OntoEdit Free	Ontoprise	http://www.ontostudio.de/
Open Ontology Forge	National Inst. of Informatics	http://research.nii.ac.jp/˜collier/resources/OOF/
Protégé 2000	Stanford University	http://protege.stanford.edu/
SemTalk	Semtation	http://www.semtalk.com/
SNOBASE	IBM	http://www.alphaworks.ibm.com/tech/snobase
Unicorn Workbench	Unicorn Solutions	http://www.unicorn.com/products/unicornsystem/workbench.htm
Visual Ontology Modeler	Sandpiper Software	http://www.sandsoft.com/products.html
WebODE	U. Politécnica de Madrid	http://webode.dia.fi.upm.es/WebODEWeb/index.html

Table 2.1: Ontology development tools capable of importing/exporting RDF(S)

Tool	Version	Developer	Experimenter
Corese	2.1.2	INRIA	INRIA
Jena	2.3	HP	U. Politécnica de Madrid
KAON	1.2.9	U. Karlsruhe	U. Karlsruhe
Sesame	2.0 alpha 3	Aduna	U. Politécnica de Madrid
Protégé	3.2 beta build 230	Stanford U.	U. Politécnica de Madrid
WebODE	2.0 build 109	U. Politécnica de Madrid	U. Politécnica de Madrid

Table 2.2: Ontology development tools participating in the benchmarking

Any tool capable of importing and exporting RDF can participate in the benchmarking. For our part, not only ontology development tools are participating in it, but also RDF repositories.

When writing this deliverable, six tools are taking part in the benchmarking, three of which are ontology development tools: KAON, Protégé (using its RDF backend), and WebODE; the other three are RDF repositories: Corese[2], Jena[3] and Sesame[4]. As Table 2.2 shows, benchmarking is not always performed by the tool developers.

Another ontology development tool, OntoStudio (the successor of OntoEdit), is also participating but it is not considered in this deliverable because its execution of the benchmark suites has not finished yet, so its complete results are not available.

2.1.7 Planning and resource allocation

The main deadline of the benchmarking was imposed by the deadline of this Knowledge Web deliverable. Therefore, a plan was designed that included the *Plan* and *Experiment* phases, though it did not include the *Improve* phase.

This plan was developed and agreed by all the organisations participating in the benchmarking; besides, every organisation had to assign a number of people to perform the benchmarking.

[2]http://www-sop.inria.fr/acacia/soft/corese/
[3]http://jena.sourceforge.net/
[4]http://www.openrdf.org/

2.2 Experiment phase

2.2.1 Experiment definition

Evaluating the interoperability of ontology development tools using RDF(S) for ontology interchange requires that the importers and exporters from/to RDF(S) of these tools work accurately when interchanging ontologies. Therefore, the planning for the experimentation includes three consecutive stages, shown in Figure 2.2:

- **Agreement stage**. The quality of the benchmark suites to be used is essential for the results of the benchmarking. Therefore, the first step is to agree on the definition of these benchmark suites, which will be common for all the tools. Chapter 3 deals with the definition and use of these benchmark suites.
- **Evaluation stage 1**. The RDF(S) importers and exporters of the ontology development tools are evaluated with the agreed versions of the benchmark suites.
- **Evaluation stage 2**. Once the RDF(S) importers and exporters are evaluated, this stage will cover the evaluation of the ontology interchange between ontology development tools.

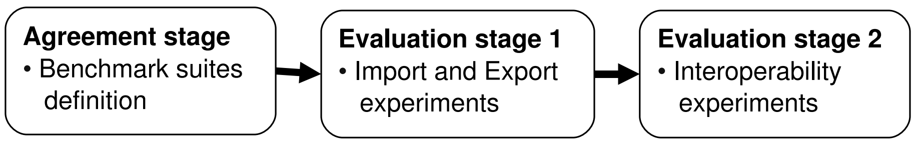

Agreement stage
• Benchmark suites definition
Evaluation stage 1
• Import and Export experiments
Evaluation stage 2
• Interoperability experiments

Figure 2.2: Experimentation Phases

2.2.2 Experiment execution

Once the benchmark suites were defined, they were published on the benchmarking web page so that they could be reviewed by the participants. Then, in Madrid, a meeting was held, the *Interoperability Working Days*[5], to discuss the benchmark suites and to improve them with comments of the participants.

In that meeting, some improvements were made to the initial benchmark suites, whereas some experiments were performed in the tools.

The actual experiments performed are the following:

[5]http://knowledgeweb.semanticweb.org/benchmarking_interoperability/working_days/

- Import, export and interoperability experiments were carried out on KAON, Protégé and WebODE.
- Import and export experiments were carried out on the RDF repositories Corese, Jena and Sesame.
- Initial import and export experiments were carried out on OntoStudio though not with the agreed versions of the benchmark suites.

In the cases of Corese, Jena, Sesame, and WebODE, most of the experimentation was automated. In the other tools, it was performed manually.

The experimentation results obtained are available in the benchmarking web page.

2.2.3 Experiment result analysis

Participants have analysed the results of the experimentation and identifyed the components that the tools can import, export and interchange. They have also identified the problems, as summarised in Chapters 5 to 6 and detailed in the appendixes.

Chapter 3

Benchmark suites definition

by RAÚL GARCÍA CASTRO

This chapter deals with the definition of the three benchmark suites used in the benchmarking and explains how they were defined. For each of the benchmark suites it includes:

- The tools and functionalities that can be evaluated with it.
- The benchmarks that compose it.
- The criteria for evaluating the benchmark results.
- The tools or data needed to run the benchmarks.
- The procedure to follow for running the benchmarks.

3.1 RDF(S) Import Benchmark Suite

The RDF(S) Import Benchmark Suite can be used to evaluate the RDF(S) import functionalities of Semantic Web tools. Although it was developed bearing in mind ontology development tools, it can be used to evaluate any other tool capable of importing RDF(S).

Each benchmark in the benchmark suite defines a RDF(S) ontology serialized in a RDF/XML file that must be loaded into the ontology development tool.

These benchmarks check the correct import of ontologies that model a simple combination of the components of the RDF(S) knowledge model (classes, properties, instances, etc.) [Brickley and Guha, 2004]. To assess the import of real, large or complex ontologies can be useless if we do not know whether the importer can deal with simple ones correctly. Because one of the goals of the benchmarking is to improve the tools, the ontologies must be simple in order to isolate problem causes and to identify possible problems.

There are two different issues that influence the correct import of an ontology. One is which combinations of components of the RDF(S) knowledge model are present in the ontology; the other is which of the different variants of the RDF/XML syntax are present in the ontology. Therefore, in order to isolate each of these issues, we have defined separately the benchmarks that depend on the RDF(S) knowledge model and those that depend on the RDF(S) syntax chosen. The next sections explain how these two types of benchmarks have been defined.

3.1.1 Benchmarks that depend on the knowledge model

These benchmarks check the correct import of RDF(S) ontologies that model simple combinations of the RDF(S) knowledge model components.

Figure 3.1 shows the different components that form the RDF(S) knowledge model and the different properties that relate them. Classes are defined as boxes in the figure, whereas properties are defined as arrows with their domain and range represented by the origin and destination of the arrow respectively. The figure does not show a full description of the knowledge model; the components shown are instances of *rdfs:Class*, and some of them have predefined instances that do not appear here.

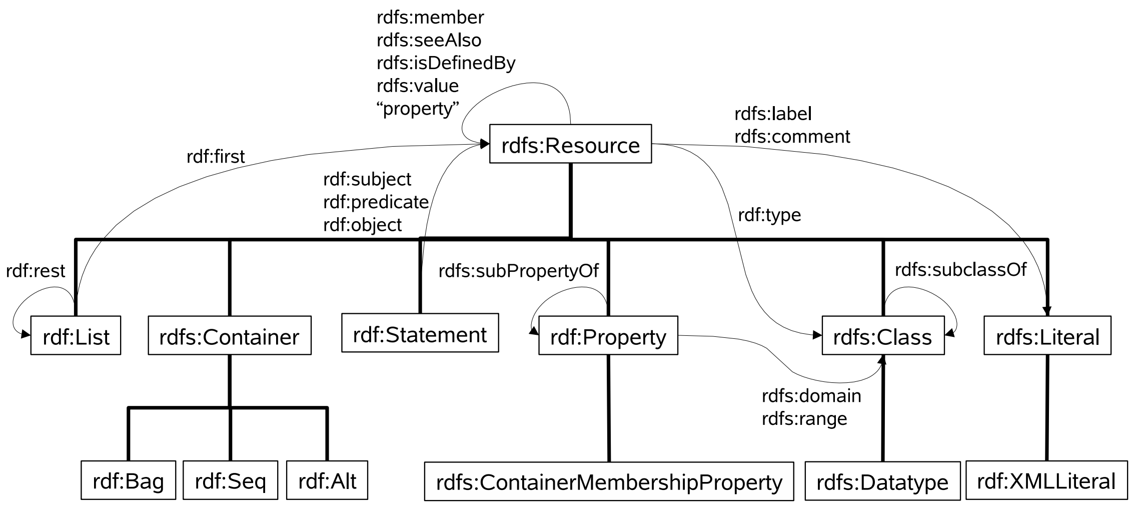


Figure 3.1: The components of the RDF(S) knowledge model

We have considered the import of all the possible combinations of the knowledge model components to make the benchmark suite exhaustive. Three different types of benchmarks depend on the knowledge model and these are:

- Benchmarks that import single components.
- Benchmarks that import all the possible combinations of two components with a property.

- Benchmarks that import combinations of more than two components usually appearing together in RDF(S) graphs.

3.1.1.1 Benchmarks that import single components

For each component of the knowledge model of RDF(S), we have defined two benchmarks that import:

- A single instance of a component.
- Several instances of a component.

3.1.1.2 Benchmarks that import combinations of two components

The method followed to define all the combinations of two components related through a property was the following. For each of the RDF(S) components:

Step 1. We have identified the possible relations of the component with others: the properties whose domain can be the component and relate it to other components. These properties are:

- RDF(S) predefined properties whose domain is the component or a superclass of the component.
- User defined properties whose domain is the component.

Step 2. For each of these relations, we identify the ranges that the property can have and we assign the cardinalities that correspond to each relation. These ranges are:

- The component defined in the RDF(S) specification as the range of the property and the components that are subclass of this component.
- If *rdfs:Class* is one of the possible ranges of the property, the RDF(S) predefined instances of *rdfs:Class*.
- An unknown component that is not defined in the rest of the RDF(S) graph, even though the component is a resource.

Step 3. The assigned cardinalities define the different number of benchmarks that will be performed for each relation as follows: for two components (c1 and c2) related through a property (p) ($c1 - p \rightarrow c2$), being the cardinality:

- 1:1 cardinality ($c1\ {}^{1}\!- p \rightarrow^{1} c2$). Define 1 benchmark to import:

  - One instance of a component related to an instance of another component through a property.

- 1:N cardinality ($c1\ {}^{1}\!- p \rightarrow^{*} c2$). Define 2 benchmarks, the one defined for the 1:1 cardinality and another to import:
  - One instance of a component related to several instances of another component through the same property.

- N:1 cardinality ($c1\ {}^{*}\!- p \rightarrow^{1} c2$). Define 2 benchmarks, the one defined for the 1:1 cardinality and another to import:
  - Several instances of a component related to an instance of another component through the same property.

- N:N cardinality ($c1\ {}^{*}\!- p \rightarrow^{*} c2$). Define 3 benchmarks, those defined for the 1:N and N:1 cardinalities.

- N:N cardinality, being c1 and c2 the same component ($c1\ {}^{*}\!- p \rightarrow^{*} c1$). Define 4 benchmarks, the three defined for the N:N cardinality and one to import:
  - One instance of a component related to itself through a property.

- N:N cardinality, being c1 and c2 the same component and p a transitive property ($c1\ {}^{*}\!- p \rightarrow^{*} c1$). Define 5 benchmarks, the four defined for the previous case and one to import:
  - One instance of a component related to an instance of another component through a property, being the second instance related to an instance of a third component with the same property.

For example, in the case of *rdfs:Class*, the following properties can relate it to other components:

- RDF(S) predefined properties whose domain is *rdfs:Class* (*rdfs:subClassOf*) or superclass of *rdfs:Class* (*rdf:type*, *rdfs:label*, *rdfs:comment*, *rdfs:member*, *rdfs:seeAlso*, *rdfs:isDefinedBy*, and *rdfs:value*).
- User defined properties whose domain is *rdfs:Class* (some fictitious property ”property”).

In the case of *rdfs:Class* with the property *rdfs:subClassOf*, the cardinalities of the relations according to the possible ranges are the following:

- The predefined range of the property (*rdfs:Class*):

  - *rdfs:Class* $^*-$ *rdfs:subClassOf* $\rightarrow^*$ *rdfs:Class*

- The subclasses of the predefined range of the property (*rdfs:Datatype*):
  - *rdfs:Class* $^*-$ *rdfs:subClassOf* $\rightarrow^*$ *rdfs:Datatype*

- The predefined instances of *rdfs:Class* (*"rdfs:Resource"*, *"rdf:Property"*, *"rdf:List"*, *"rdfs:Datatype"*, *"rdfs:Class"*, *"rdfs:Container"*, *"rdf:Bag"*, *"rdf:Seq"*, *"rdf:Alt"*, *"rdfs:ContainerMembershipProperty"*, and *"rdf:Statement"*):
  - *rdfs:Class* $^*-$ *rdfs:subClassOf* $\rightarrow^1$ *"rdfs:Resource"*
  - *rdfs:Class* $^*-$ *rdfs:subClassOf* $\rightarrow^1$ *"rdfs:Datatype"*
  - ...

- An unknown component:
  - *rdfs:Class* $^*-$ *rdfs:subClassOf* $\rightarrow^*$ *"unknown"*

For the relation *rdfs:Class* $^*-$ *rdfs:subClassOf* $\rightarrow^*$ *"rdfs:Class"*, we can define 5 different benchmarks to import:

- One class that is a subclass of another.
- One class that is a subclass of several other classes.
- Several classes that are subclasses of another class.
- One class that is a subclass of itself.
- One class that is a subclass of another, being the second one a subclass of a third one.

3.1.1.3 Benchmarks that import combinations of more than two components

We have identified the main combinations of RDF(S) components that involve more than two components related through properties. These combinations are:

- Properties that have both domain and range (*rdf:Property* with *rdfs:domain* and *rdfs:range*).
- Statements that have subject, predicate and object (*rdf:Statement* with *rdf:subject*, *rdf:predicate* and *rdf:object*).
- Definitions of lists (*rdf:List* with *rdf:first*, *rdf:rest* and *rdf:nil*).

The method to define the benchmarks is similar to the one described in the previous section. The main difference resides in the number of benchmarks defined according to the cardinalities. To clarify the explanation below, we are going to consider a cardinality of 1 in the origin of the relation.

- If the cardinalities of the relations are
  $c1\ {}^{1}\!-\ p1 \rightarrow^{+} c2$
  $c1\ {}^{1}\!-\ p2 \rightarrow^{+} c3$
  four benchmarks have been defined to import:
  - One instance of a component related to an instance of another component through a property and related to an instance of a third component through another property.
  - One instance of a component related to several instances of another component through a property and related to an instance of another component through another property.
  - One instance of a component related to an instance of another component through a property and related to several instances of another component through another property.
  - One instance of a component related to several instances of another component through a property and related to several instances of another component through another property.

  If c2 and c3 are the same component, an additional benchmark has been defined to import:
  - One instance of a component related to an instance of another component through the two properties.
- If the cardinalities of the relations are
  $c1\ {}^{1}\!-\ p1 \rightarrow^{1} c2$
  $c1\ {}^{1}\!-\ p2 \rightarrow^{+} c3$
  or
  $c1\ {}^{1}\!-\ p1 \rightarrow^{+} c2$
  $c1\ {}^{1}\!-\ p2 \rightarrow^{1} c3$
  two benchmarks have been defined to import:
  - One instance of a component related to an instance of another component through a property and also related to an instance of a third component through another property.
  - One instance of a component related to an instance of another component through a property and also related to several instances of another component through another property.

- If the cardinalities of the relations are
  c1 ^{1}– *p1* $\rightarrow^1$ *c2*
  c1 ^{1}– *p2* $\rightarrow^1$ *c3*
  one benchmark has been defined to import:
  - One instance of a component related to an instance of another component through a property and also related to an instance of a third component through another property.

For example, for a property with domain and range and with the following ranges and cardinalities:
rdf:Property ^{1}– *rdfs:domain* $\rightarrow^+$ *rdfs:Class*
rdf:Property ^{1}– *rdfs:range* $\rightarrow^+$ *rdfs:Class*
5 benchmarks have been defined to import:

- One property that has as domain a class and as range another class.
- One property that has as domain a class and as range several classes.
- One property that has as domain several classes and as range another class.
- One property that has as domain several classes and as range other several classes.
- One property that has as domain and range the same class.

3.1.1.4 Pruning the benchmark suite

As RDF(S) does not impose any restriction on the combinations of its components, the number of the resulting benchmarks is huge (more than 4000) and the benchmark suite has to be pruned according to its intended use and the kind of tools that it is supposed to evaluate, namely, ontology development tools. Therefore, we have only considered the RDF(S) components that can be used for modelling ontologies in all these tools: *rdfs:Class*, *rdf:Property*, *rdfs:Literal*, *rdf:type*, *rdfs:domain*, *rdfs:range*, *rdfs:subClassOf*, and *rdfs:subPropertyOf*. The rest of the RDF(S) components have not been dealt with.

The benchmarks obtained are classified in the following categories:

- Class benchmarks
- Metaclass benchmarks
- Subclass benchmarks
- Class and property benchmarks
- Single property benchmarks

- Subproperty benchmarks
- Property with domain and range benchmarks
- Instance benchmarks
- Instance and property benchmarks

3.1.2 Benchmarks that depend on the RDF/XML syntax

These benchmarks check the correct import of RDF(S) ontologies with the different variants of the RDF/XML syntax, as stated in the RDF/XML specification.

We have defined benchmarks that take into account the following variants of the RDF/XML syntax:

- Different syntax of URI references:
  - Absolute URI references
  - URI references relative to a base URI
  - URI references transformed from *rdf:ID* attribute values
  - URI references relative to an ENTITY declaration
- Language identification attributes (*xml:lang*) in tags.
- Empty node abbreviations.
- Multiple properties abbreviations.
- Typed node abbreviations.
- String literal abbreviations.
- Blank node abbreviation.
- Container abbreviation.
- Collection abbreviation.
- Statement abbreviation.

3.1.3 Benchmark definitions

Table 3.1 shows the categories of the RDF(S) Import Benchmark Suite, which contains 82 benchmarks, their number and the components used in each category. A detailed description of these benchmarks can be found in the web page[1]. Besides, all the RDF(S) files to be imported can be downloaded from a single file[2]. Templates are provided for collecting the execution results[3].

Table 3.1: Categories of the import benchmarks.

Category	**No.**	**Components used**
Class	2	*rdfs:Class*
Metaclass	5	*rdfs:Class*, *rdf:type*
Subclass	5	*rdfs:Class*, *rdfs:subClassOf*
Class and property	6	*rdfs:Class*, *rdf:Property*, *rdfs:Literal*
Property	2	*rdf:Property*
Subproperty	5	*rdf:Property*, *rdfs:subPropertyOf*
Property with domain and range	24	*rdfs:Class*, *rdf:Property*, *rdfs:Literal*, *rdfs:domain*, *rdfs:range*
Instance	4	*rdfs:Class*, *rdf:type*
Instance and property	14	*rdfs:Class*, *rdf:type*, *rdf:Property*, *rdfs:Literal*
Syntax and abbreviation	15	*rdfs:Class*, *rdf:type*, *rdf:Property*, *rdfs:Literal*

The definition of each benchmark in the benchmark suite, as Table 3.2 shows, includes the following fields:

- An **identifier** for tracking the different benchmarks.
- A **description** of the benchmark in natural language.
- A **graphical representation** of the ontology to be imported in the benchmark.
- A **file** containing the ontology to be imported in the RDF/XML syntax.

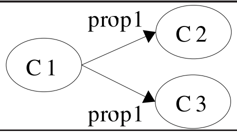
3.1.4 Evaluation criteria

The evaluation criteria of the RDF(S) Import Benchmark Suite are defined as follows:

[1] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/rdfs_import_benchmark_suite.html

[2] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/files/import_files.zip

[3] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/templates/RDFS_Import_Benchmark_Suite_Template.xls

Identifier	I14
Description	Import one class that has the same property with several other classes
Graphical representation	

RDF/XML file	`<rdf:RDF xmlns="http://www.w3.org/2000/01/rdf-schema#"`
	` xmlns:g1="http://www.test.org/graph14#"`
	` xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"`
	` xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">`
	` <Class rdf:about="http://www.test.org/graph14#class1">`
	`  <g1:prop1 rdf:resource="http://www.test.org/graph14#class2"/>`
	`  <g1:prop1 rdf:resource="http://www.test.org/graph14#class3"/>`
	` </Class>`
	` <Class rdf:about="http://www.test.org/graph14#class2"/>`
	` <Class rdf:about="http://www.test.org/graph14#class3"/>`
	`</rdf:RDF>`

Table 3.2: An example of a benchmark definition.

Modelling (YES/NO). The ontology development tool can model the ontology components described in the benchmark.

Execution (OK/FAIL). The execution of the benchmark is normally carried out without any problem, and the tool always produces its expected result. In the case of a failed execution, the following information is required:

- The reasons for failing the benchmark execution.
- If the tool was corrected to pass a benchmark, which it underwent.

Information added or lost. It refers to the information added or lost in the ontology interchange when executing the benchmark.

Since ontology development tools have different knowledge models, there is no *Right* or *Wrong* result. Furthermore, different tools have different strategies for importing the components not allowed in their knowledge models. For example, metaclasses can be modelled in RDF(S), but a tool that cannot represent metaclasses has two alternatives when importing a RDF(S) metaclass: either to import it as a class, or not to do it.

In addition, any combination of results can be possible since they depend on the decisions taken by the tool developers. The only pattern that can be identified in the results is that of the loss of information during the import of an ontology with a component that does not belong to its knowledge model. This loss of information includes at least the component that the tool cannot model.

Table 3.3 shows an example of the execution of the benchmark I46 (*Import just one property that has as domain a class and as range the XML Schema datatype "string", with the class defined in the ontology*) in five fictitious ontology development tools identified as A, B, C, D, and E.

Tool	**ID**	**Modelling**	**Execution**	**Information added**	**Information lost**
A	I46	YES	OK	A label in all the components	-
B	I46	YES	FAIL	-	The property's range
C	I46	NO	OK	The range *String*	The range *xsd:string*
D	I46	NO	OK	The range *rdfs:Literal*	The range *xsd:string*
E	I46	NO	FAIL	-	The property

Table 3.3: Fictitious results of executing benchmark I46

In our example, tools A and B can model the XML Schema datatype *string* as range and, therefore, their *Modelling* result is *YES*; tools C, D and E cannot model it and, therefore, their *Modelling* result is *NO*.

The expected result of tools A and B is a property whose domain is a class and whose range is the XML Schema datatype *string*. Tool A imports all these components and adds a label with the name of the component to all the components; therefore, its *Execution* result is *OK*, and it inserts new information into the ontology. Tool B imports the property, but it does not import the range. As it does not produce the expected result, its *Execution* result is *FAIL*, and it loses information when importing the ontology.

Because tools C, D and E cannot model the XML Schema datatype *string* as range though they can model string ranges, the expected result of these tools is to have a property whose domain is a class and whose range is string. Tools C and D produce this expected result and their *Execution* result is *OK*; both lose information about the range being the XML Schema datatype *string*, though tool C creates the range as its own datatype *String* and tool D creates the range as *rdfs:Literal*; therefore, these two tools insert new information in the ontology. Tool E does not import the property at all, although its expected result is to import it with a string range; its *Execution* result is *FAIL*, and it loses all the information about the property when it imports the ontology.

3.1.5 Procedure for executing the benchmark suite

If a tool developer wants to evaluate the RDF(S) importer of his tool, the steps he should follow for executing each benchmark are:

1. To define the expected result of importing the file with the RDF(S) ontology into the ontology development tool, either by modelling the expected ontology in the tool or by defining it informally (i.e. in natural language).

2. To import into the ontology development tool the RDF(S) file that contains the RDF(S) ontology defined in the benchmark.

3. To compare the expected ontology with the imported one and to check whether there is some addition or loss of information.

Although these steps can be carried out manually, when dealing with many benchmarks it is highly recommended to perform them (or part of them) automatically, especially to compare the expected with the imported ontologies.

3.2 RDF(S) Export Benchmark Suite

The RDF(S) Export Benchmark Suite can be used to evaluate the RDF(S) export functionalities of Semantic Web tools. Although it was developed bearing in mind ontology development tools, it can also be employed to evaluate any other tool capable of exporting to RDF(S).

The benchmark suite is composed of benchmarks that check the correct export of ontologies to RDF(S). Each of these benchmarks defines an ontology that must be modelled in the ontology development tool and saved to a RDF(S) file.

As in the case of the RDF(S) Import Benchmark Suite, we have defined two types of benchmarks for isolating the two issues that influence the correct exporting of an ontology. One is which combinations of components of the ontology development tool knowledge model are present and the other issue is which restrictions RDF(S) imposes for naming components. The next sections describe how these two types of benchmarks were defined.

3.2.1 Benchmarks that depend on the knowledge model

These benchmarks check the correct export to RDF(S) of ontologies that model simple combinations of the components of a common knowledge model. The components here considered are the most frequently used for modelling ontologies in these tools, and are present in their knowledge models. These are: classes and class hierarchies, object and datatype properties, instances, and literals; the remainder of the components that are specific to each tool are not considered.

The composition of the RDF(S) Export Benchmark Suite is similar to the composition of the Import one. Instead of taking as input the knowledge model of RDF(S), we have taken the common core of knowledge modelling components mentioned above. The benchmarks obtained are classified in the following categories:

- Class benchmarks
- Metaclass benchmarks
- Subclass benchmarks

- Class and object property benchmarks
- Class and datatype property benchmarks
- Object property benchmarks
- Datatype property benchmarks
- Instance benchmarks
- Instance and object property benchmarks
- Instance and datatype property benchmarks

3.2.2 Benchmarks that depend on the component naming restrictions

These benchmarks check the correct export to RDF(S) of ontologies with concepts and properties whose names include characters not allowed for representing RDF(S) or XML URIs. They are classified in the following categories:

- Concepts and properties whose names start with a character other than a letter or ’_’.
- Concepts and properties with spaces in their names.
- Concepts and properties with URI reserved characters in their names (’;’, ’/’, ’?’, ’:’, ’@’, ’&’, ’=’, ’+’, ’$’, ’,’).
- Concepts and properties with XML delimiter characters in their names (’¡’, ’¿’, ’#’, ’%’, ’ ”’).

3.2.3 Benchmark definitions

Table 3.4 shows the categories of the benchmark suite, which comprises 66 benchmarks. The table contains the number of benchmarks and the components used in each category. A detailed description of such benchmarks can be found in the web page[4]. In addition, templates are provided for collecting the execution results[5].

The definition of each benchmark in the benchmark suite, as Table 3.5 shows, includes the following fields:

[4]http://knowledgeweb.semanticweb.org/benchmarking_interoperability/rdfs_export_benchmark_suite.html

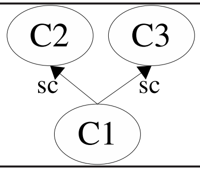
[5]http://knowledgeweb.semanticweb.org/benchmarking_interoperability/templates/RDFS_Export_Benchmark_Suite_Template.xls

Table 3.4: Categories of the export benchmarks.

Category	**No.**	**Components used**
Class	2	*class*
Metaclass	5	*class*, *instanceOf*
Subclass	5	*class*, *subClassOf*
Class and object property	4	*class*, *object property*
Class and datatype property	2	*class*, *datatype property*, *literal*
Object property	14	*object property*
Datatype property	12	*datatype property*
Instance	4	*class*, *instanceOf*
Instance and object property	9	*class*, *instanceOf*, *object property*
Instance and datatype property	5	*class*, *instanceOf*, *datatype property*, *literal*
URI character restrictions	4	*class*, *instanceOf*, *object property*, *datatype property*, *literal*

- An **identifier** for tracking the different benchmarks.
- A **description** of the benchmark in natural language.
- A **graphical representation** of the ontology to be exported by the tool.
- The **instantiation** of the ontology described in the benchmark for each of the participating tools, using the vocabulary and components of these tools.

Table 3.5: An example of a benchmark definition.

Identifier	E09
Description	Export one class that is subclass of several classes
Graphical representation	
WebODE's instantiation	Export one concept that is subclass of several concepts
Protégé's instantiation	Export one class that is subclass of several classes
...	...

3.2.4 Evaluation criteria

The evaluation criteria adopted for the export benchmark suite are the same as those for the import benchmark suite, namely, *Modelling*, *Execution* and *Information added or lost*. The only difference with the import criteria is that, if a benchmark defines an ontology that cannot be modelled in a certain tool, that benchmark cannot be executed in the tool, being the *Execution* result *N.E.* (Non Executed). In the import benchmark suite, even if a tool cannot model some components of the ontology, it should be able to import correctly the rest of the components.

3.2.5 Procedure for executing the benchmark suite

The steps to follow when executing each of the benchmarks are:

1. To define in RDF(S) the expected ontology that results from exporting the ontology.
2. To model in the tool the ontology described in the benchmark.
3. To export the ontology modelled using the tool to RDF(S).
4. To compare the exported RDF(S) ontology with the expected RDF(S) ontology, examining whether there is some addition or loss of information.

Although these steps can be carried out manually, when dealing with many benchmarks it is highly recommended to perform them (or part of them) automatically, especially for comparing the expected with exported ontologies.

3.3 RDF(S) Interoperability Benchmark Suite

The RDF(S) Interoperability Benchmark Suite can be used to evaluate the interoperability of Semantic Web tools using RDF(S) as interchange language. It does so by testing the interchange of ontologies from one source tool to a destination one and vice versa. Although it was developed bearing in mind ontology development tools, it can be used to evaluate any other tool capable of importing from and exporting to RDF(S).

The RDF(S) Interoperability Benchmark Suite is composed of benchmarks that check the correct interchange of ontologies between two tools. The benchmark suite considers the interchange of a common core of the knowledge modelling components most frequently used for modelling ontologies: classes and class hierarchies, object and datatype properties, instances, and literals. As these components are the same as those in the RDF(S) Export Benchmark Suite, the ontologies defined in the RDF(S) Interoperability Benchmark Suite are identical to those of the RDF(S) Export Benchmark Suite, as presented in the previous section.

The RDF(S) Interoperability Benchmark Suite definition is available in a public web page[6] and templates are provided for collecting the execution results[7]. In case of evaluating the interoperability from the tools that have already executed the RDF(S) Export Benchmark Suite, a file containing the RDF(S) files exported by these tools can be downloaded[8].

The evaluation criteria adopted here are the same as those for the export benchmark suite, namely, *Modelling*, *Execution* and *Information added or lost*.

3.3.1 Procedure for executing the benchmark suite

The main difference between the RDF(S) Export and Interoperability Benchmark Suites is the procedure for executing the benchmarks. The steps to follow for executing them are:

1. To define the expected ontology resulting from interchanging the ontology in the destination tool.
2. To model the ontology described in the benchmark in the source tool.
3. To export the ontology modelled using the source tool to RDF(S).
4. To import the RDF(S) file (exported by the source tool) into the destination tool.
5. To compare the interchanged ontology with the expected one, checking whether there is some addition or loss of information.

If the tools have already executed the RDF(S) Export Benchmark Suite, then steps 2 and 3 can be ignored, as the RDF(S) exported files of all the tools will be available from the export experiments. Participants will not have to export these ontologies again; they will only have to import the exported files into their tools.

Although these steps can be carried out manually, it is highly recommended to perform them (or part of them) automatically when dealing with many benchmarks, especially for comparing the expected with the interchanged ontologies.

[6]http://knowledgeweb.semanticweb.org/benchmarking_interoperability/rdfs_interoperability_benchmark_suite.html

[7]http://knowledgeweb.semanticweb.org/benchmarking_interoperability/templates/Interoperability Templates.xls

[8]http://knowledgeweb.semanticweb.org/benchmarking_interoperability/stage_1_results/RDFS Exported Files.zip

Chapter 4

RDF(S) import results and analysis

by RAÚL GARCÍA-CASTRO

This chapter presents the results of executing the RDF(S) Import Benchmark Suite in all the tools participating in the benchmarking. First, we carry out a global analysis on the RDF(S) import capabilities of the tools and then an in-depth analysis of each tool is provided.

The results obtained when importing from RDF(S) depend mainly on the knowledge model of the tool that executed the benchmark suite. The tools that natively support the RDF(S) knowledge model do not need to perform any translation in the ontologies when importing them from RDF(S). In the case of tools with non-RDF knowledge models, they do need to translate ontologies from RDF(S) to their knowledge model.

In the benchmarking, the only RDF(S)-native participant tools are Corese, Jena and Sesame, the RDF repositories. The ontology development tools (KAON, Protégé and WebODE) have their own knowledge models, some of whose components can also be represented in RDF(S) while some others cannot.

The RDF repositories import correctly from RDF(S) all the combinations of components, as the import does not require any translation.

In general, the ontology development tools import correctly from RDF(S) most of the combinations of components that they model, rarely adding or losing information. In particular:

- KAON imports correctly all the combinations of components that it can model.
- Protégé only poses problems when importing classes or instances that are instances of multiple classes.
- WebODE only poses problems when importing properties with a XML Schema datatype as range.

When the ontology development tools import ontologies with combinations of components that they cannot model, they lose the information about these components. Nevertheless, they usually try to represent partially these components using other components from their knowledge models. In most cases, the importing is performed correctly. The only exceptions are:

- KAON poses problems when it imports class hierarchies with cycles.
- Protégé poses problems when it imports class and property hierarchies with cycles and properties with multiple domains.
- WebODE poses problems when it imports properties with multiple domains or ranges.

When dealing with the different variants of the RDF/XML syntax, ontology development tools:

- Import correctly resources with the different URI reference syntaxes.
- Import correctly resources with the different syntaxes (shortened and unshortened) of empty nodes, of multiple properties, of typed nodes, of string literals, and of blank nodes. The only exceptions are KAON when it imports resources with multiple properties in the unshortened syntax, and Protégé when it imports resources with empty and blank nodes in the unshortened syntax.
- Do not import language identification attributes (*xml:lang*) in tags.

The next sections present a summary of the analysis of each of the tools. A detailed analysis of the RDF(S) import results can be found in Appendix B.

4.1 KAON results

by YORK SURE AND MARKUS ZONDLER

This section presents the results of evaluating the RDF(S) import capabilities of KAON.

KAON imports correctly components (or combinations of them) of the RDF(S) knowledge model, which are also present in its own knowledge model. These components are:

- Classes, metaclasses and class hierarchies without cycles.
- Properties with a single domain and range (even if the domain and range are the same class), with multiple domains or ranges, or without domain or range.
- Property hierarchies.

- Instances of one class or multiple classes and instances related through properties, even if the property relates instances of the same class or an instance to itself, or if an instance is related through the same property (either as subject or object) to several other instances or values.

KAON imports correctly the following components (or combinations of them) of the RDF(S) knowledge model, although they are not present in its own knowledge model:

- Classes related through properties supposed to be defined with a domain and range of some metaclass of the classes.
- Properties with undefined resources as domain or range or with a XML Schema datatype as range.
- Instances of undefined resources.
- Instances related through undefined properties or through properties whose range is a XML Schema datatype.

KAON does not produce the expected result when importing from RDF(S):

- Class hierarchies with cycles because KAON presents an error message.

Regarding the import of the different variants of the RDF/XML syntax, KAON:

- Imports correctly resources with the different URI reference syntaxes.
- Imports correctly resources with the different syntaxes (shortened and unshortened) of empty nodes, typed nodes, string literals, and blank nodes.
- Imports correctly resources with multiple properties in the shortened syntax, but it crashes if they are in the unshortened.
- Does not import language identification attributes (*xml:lang*) in tags.

4.2 Protégé results

by RAÚL GARCÍA-CASTRO

This section presents the results of evaluating the RDF(S) import capabilities of Protégé.

Protégé imports correctly the following components (or combinations of them) of the RDF(S) knowledge model, which are also present in its own knowledge model:

- Classes, metaclasses and classes that are instances of a single metaclass.

- Class hierarchies without cycles. If the resource object of a *rdfs:subClassOf* property is not defined as a class in the file ([*Corese:In09-10*]), the resource object is created as a class.
- Properties with a single domain and range (even if the domain and range are the same class) or without domain or range, or without both.
- Property hierarchies without cycles.
- Instances of a single class and instances related through properties, even if the property relates instances of the same class or an instance to itself, or if an instance is related through the same property (either as subject or object) to several other instances or values.

Protégé imports correctly components (or combinations of them) of the RDF(S) knowledge model, although they are not present in its own knowledge model. These components are:

- Classes related through properties supposed to be defined with a domain and a range of some metaclass of the classes. Protégé imports the property, but without domain or range, and it does not relate the classes to the property.
- Properties with undefined resources as domain or range. Protégé creates the undefined resource as a class.
- Properties with multiple ranges. Protégé creates the slot with a range of *Any*, as multiple ranges in RDF(S) and Protégé have different meanings.
- Properties with *rdfs:Class* as domain. Protégé does not define the domain as it cannot create a template slot in *:STANDARD_CLASS*.
- Properties with a XML Schema datatype as range. It creates the datatype as a class with the *xsd* namespace.
- Instances of undefined resources. Protégé creates the undefined resource as a class.
- Instances related through undefined properties. Protégé creates the property as a template slot without domain and with a range of *Any* and it does not relate the instances to the property.
- Instances related through properties with XML Schema datatype as range. Protégé creates the datatype as a class with the *xsd* namespace, but it does not consider the value related to the instance through the property to be an instance of the datatype class and it does not import it.
- *rdfs:Class*. Protégé imports *rdfs:Class* as *:STANDARD_CLASS*.

- *rdfs:Literal.* Protégé imports *rdfs:Literal* as its own *String* datatype.

Protégé does not produce the expected result when importing from RDF(S):

- Classes that are instances of multiple metaclasses because Protégé imports the class as instance of only one metaclass. This includes the case in which a class appears in the file as instance of a metaclass and *rdfs:Class* ([*Corese:In03-07*],[*KAON:In03-07*]).
- Class hierarchies with cycles because Protégé crashes when finding a cycle in a class hierarchy, so it does not importing anything.
- Properties with multiple domains because Protégé imports the multiple domains, but in RDF(S) and in Protégé multiple domains have different meanings.
- Property hierarchies with cycles because Protégé crashes when finding a cycle in a property hierarchy, so it does not import anything.
- Instances of multiple classes because Protégé imports the instance as instance of only one class.

Regarding the import of the different variants of the RDF/XML syntax, Protégé:

- Imports correctly resources with the different URI reference syntaxes.
- Imports correctly resources with the different syntaxes (shortened and unshortened) of multiple properties, typed nodes, and string literals.
- Imports correctly resources with empty nodes in the shortened syntax, but Protégé crashes when they are in the unshortened syntax, so it does not import anything.
- Imports correctly resources with blank nodes in the shortened syntax. However, if they are in the unshortened syntax, whenever the blank node appears it is imported as a new node.
- Does not import language identification attributes (*xml:lang*) in tags.

4.3 WebODE results

by RAÚL GARCÍA-CASTRO

This section includes the results of evaluating the RDF(S) import capabilities of WebODE.

WebODE imports correctly components (or combinations of them) of the RDF(S) knowledge model that are also present in its own knowledge model. These components are:

- Classes and class hierarchies without cycles.
- Properties with a single domain and range, even if the domain and range are the same class.
- Instances of a single class and instances related through properties, even if the property relates instances of the same class or relates an instance to itself, or if an instance is related through the same property (either as subject or object) to several other instances or values.

WebODE imports correctly the following components (or combinations of them) of the RDF(S) knowledge model although they are not present in its own knowledge model:

- Metaclasses. WebODE imports them as concepts (if they are defined as classes in the file) but it does not import that another concept is instance of them.
- Class hierarchies with cycles. WebODE does not import all the *rdfs:subClassOf* properties to create a hierarchy without cycles.
- Classes related through properties supposed to be defined with a domain and range of some metaclass of the classes. WebODE does not import the property.
- Properties without domain. WebODE creates *rdfs:Resource* as an imported term; if the range of the property is *rdfs:Literal*, it creates the property as an instance attribute of *rdfs:Resource*; otherwise, it creates the property as a relation with *rdfs:Resource* as origin.
- Properties without range. WebODE creates *rdfs:Resource* as an imported term and creates the property as a relation with *rdfs:Resource* as destination.
- Property hierarchies. WebODE does not import the *rdfs:subPropertyOf* properties.
- Properties with undefined resources as domain or range. WebODE creates the undefined resource as a concept.
- Instances of undefined resources. WebODE does not import the instance.
- Instances of multiple classes. WebODE imports the instance as instance of just one concept.
- Instances related through undefined properties. WebODE does not create the property and does not relate the instances to the property.
- *rdfs:label* properties in classes, properties and instances. WebODE inserts ”rdfs:label -> name” in the description of the resource.
- *rdfs:Class*. WebODE imports *rdfs:Class* as an imported term.

- *rdfs:Literal.* WebODE imports *rdfs:Literal* as its own datatype *String*.

WebODE does not produce the expected result when importing from RDF(S):

- Properties with multiple domains. WebODE creates an anonymous concept as the origin of the relation (or instance attribute) and as subclass of one of the domain concepts, instead of as subclass of all the domain concepts.
- Properties with multiple ranges. WebODE creates an anonymous concept as the destination of the relation and as subclass of one of the range concepts, instead of as subclass of all the range concepts.
- Properties with XML Schema datatype as range. WebODE imports the datatype as an imported term, instead of as a XML Schema datatype.
- Properties whose range is a XML Schema datatype and instances with values in the properties. WebODE inserts its own datatype *String* as the type of the instance attribute instead of the XML Schema datatype.

Regarding the import of the different variants of the RDF/XML syntax, WebODE:

- Imports correctly resources with the different URI reference syntaxes.
- Imports correctly resources with the different syntaxes (shortened and unshortened) of empty nodes, multiple properties, typed nodes, string literals, and blank nodes.
- Does not import language identification attributes (*xml:lang*) in tags.

4.4 RDF repositories: Corese, Jena and Sesame

by OLIVIER CORBY, RAÚL GARCÍA-CASTRO AND JESÚS PRIETO-GONZÁLEZ

This section presents the results of evaluating the RDF(S) import capabilities of Corese, Jena and Sesame, the RDF repositories that participated in the benchmarking.

The procedure for executing the RDF(S) Import Benchmark Suite in the RDF repositories was different from the one proposed in the benchmark suite. This is so because some tasks proposed in the benchmark suite require functionalities that are available in ontology development tools but not in RDF repositories, such as modelling the ontology into the tool.

The procedure followed in the RDF repositories was the following:

- To load the file with the RDF(S) ontology into the tool.
- To export the loaded ontology to a file using a generic query.

- To compare the expected ontology with the imported one, checking whether there is some addition or loss of information.

The RDF repositories import correctly all the combinations of components. This is so because RDF is the knowledge model of these tools, and importing ontologies to RDF does not require any transformation.

Chapter 5

RDF(S) export results and analysis

by RAÚL GARCÍA-CASTRO

This chapter includes the results of executing the RDF(S) Export Benchmark Suite in all the tools participating in the benchmarking. For this, a global analysis of the tools on the RDF(S) export capabilities has been performed; in the following sections, a specific analysis of each tool is provided.

As with the import results, the export results also depend on the knowledge model of the tool. The tools that natively support the RDF(S) knowledge model (Corese, Jena and Sesame) do not need to perform any translation when exporting ontologies, whereas the non-RDF tools (KAON, Protégé and WebODE) do.

The RDF repositories export correctly all the combinations of components to RDF(S), as the export does not require any translation.

In general, the ontology development tools export correctly to RDF(S) most of the combinations of components that they model without losing information. In particular:

- KAON poses problems only when exporting to RDF(S) datatype properties without range and datatype properties with multiple domains and a XML Schema datatype as range.

- Protégé poses problems only when exporting to RDF(S) classes or instances that are instances of multiple classes and template slots with multiple domains.

- WebODE exports correctly to RDF(S) all the combinations of components.

When ontology development tools export components present in their knowledge model but which cannot be represented in RDF(S), such as their own datatypes, they usually insert new information in the ontology though they lose some information.

When dealing with concepts and properties whose names do not fulfil URI character restrictions, each ontology development tool behaves differently:

- When names do not start with a letter or "_", some tools leave the name unchanged and some replace the first character with "_".
- Spaces in names are replaced by "-" or "_", depending on the tool.
- URI reserved characters and XML delimiter characters are left unchanged, replaced by "_", or encoded, depending on the tool.

The next sections present a summary of the analysis of each of the tools. A detailed analysis of the RDF(S) export results can be found in Appendix A.

5.1 KAON results

by YORK SURE AND MARKUS ZONDLER

This section includes the results of evaluating the RDF(S) export capabilities of KAON.

KAON exports correctly to RDF(S) some components (or combinations of them) of its knowledge model, which are also present in the RDF(S) knowledge model. These are:

- Classes, metaclasses and class hierarchies without cycles.
- Datatype properties with a domain and range, without domain and with range, or with multiple domains.
- Object properties with or without domain and range, or with multiple domains or ranges.
- Instances.
- Instances related through object properties, even if the property relates instances of the same class or relates an instance to itself, or if an instance is related through the same property (either as subject or object) to several other instances.
- Instances related through datatype properties, even if an instance is related through the same property to several values.

Although some components (or combinations of them) are not present in the RDF(S) knowledge model, KAON exports correctly the following components of its knowledge model:

- Datatypes *String* and *Integer*. KAON exports *String* and *Integer* datatypes by modelling extra concepts in the ontology or by importing them from other ontologies.

KAON does not produce the expected result when exporting to RDF(S):

- Datatype properties without range because it inserts *rdfs:Literal* as the range of the datatype property.
- Datatype properties with multiple domains and XML Schema datatype as range because the exported property contains just one domain.

KAON cannot model the following components (or combinations of them) that appear in the benchmark definitions:

- Class hierarchies with cycles.
- Classes related through undefined object or datatype properties whose domain and range are some metaclass of the classes.
- Object and datatype properties with undefined resources as domain or range.
- Instances of undefined resources.
- Instances related through undefined object or datatype properties.
- Instances related through datatype properties whose range is a XML Schema datatype.

Regarding the export of concepts and properties whose names include URI character restrictions, KAON:

- Does not modify the name of a concept, nor of an instance attribute, nor of a relation when it does not start with a letter or ”_”.
- Encodes spaces in concept names, in instance attribute names, and in relation names as ”-”.
- Encodes URI reserved characters and XML delimiter characters in class and property names.

5.2 Protégé results

by RAÚL GARCÍA-CASTRO

This section presents the results of evaluating the RDF(S) export capabilities of Protégé.

Protégé exports correctly to RDF(S) components (or combinations of them) of its knowledge model, also present in the RDF(S) knowledge model. These components are:

- Classes, metaclasses, and class hierarchies without cycles.
- Classes and instances that are instances of one class only.

- Template slots, with or without domain or range.
- Instances related through template slots, even if the slot relates instances of the same class, relates an instance to itself, or an instance is related through the same slot (either as subject or object) to several other instances.

Protégé exports correctly the following components (or combinations of them) of its knowledge model, although they are not present in the RDF(S) knowledge model:

- Resource names. Protégé inserts a *rdfs:label* property with the name of the resource when exporting classes, template slots and instances.
- Protégé datatype *String.* Protégé exports its own *String* datatype to *rdfs:Literal*.
- Exports classes as subclass of *rdfs:Resource*.
- Exports metaclasses as subclass of *rdfs:Class*.
- Exports template slots whose range is *Instance* with no allowed class or template slots with multiple ranges as properties with a range of *rdfs:Resource*.

Protégé does not produce the expected result when exporting to RDF(S):

- Classes or instances that are instances of multiple classes because it only exports the resource as instance of one class.
- Template slots with multiple domains. Protégé exports the multiple domains but in RDF(S) and in Protégé multiple domains have different meaning.

Protégé cannot model the following components (or combinations of them) that appear in the benchmark definitions:

- Class hierarchies with cycles.
- Classes related through undefined template slots whose domain and range are some metaclass of the classes.
- Template slots with undefined resources as domain or range.
- Template slots with XML schema datatype as range.
- Instances of undefined resources.
- Instances related through undefined template slots.

With regard to the export of concepts and properties whose names include URI character restrictions, Protégé:

- Inserts ”_” as the first character of the name of a class or a template slot when it does not start with a letter or ”_”.
- Encodes spaces, most URI reserved characters, and XML delimiter characters in class and property names as ”_”.

5.3 WebODE results

by RAÚL GARCÍA-CASTRO

This section presents the results of evaluating the RDF(S) export capabilities of WebODE.

WebODE exports correctly to RDF(S) components (or combinations of them) of its knowledge model that are also present in the RDF(S) knowledge model. These components are:

- Concepts and concept hierarchies without cycles.
- Instance attributes of a concept with either a WebODE datatype or a XML Schema datatype.
- Relations between two concepts or a concept and itself.
- Instances of only one concept.
- Instances related through relations, even if the relation is held between instances of the same concept or between an instance with itself, or if an instance has the same relation (either as origin or destination) to several other instances.
- Instances with instance attributes, even if an instance has several values in the instance attribute, or if the instance attribute has an XML Schema datatype as type.

WebODE exports correctly components (or combinations of them) of its knowledge model, although they are not present in the RDF(S) knowledge model. These components are:

- WebODE datatypes. WebODE exports all its own datatypes (including *String)* to *rdfs:Literal*.
- Resource names. WebODE inserts a *rdfs:label* property with the name of the resource when exporting concepts, instance attributes, relations, and instances.

WebODE produces the expected result in all the benchmarks when exporting to RDF(S).

WebODE cannot model components (or combinations of them) that appear in the benchmark definitions. These components are:

- Metaconcepts.
- Concept hierarchies with cycles.
- Concepts related through undefined relations whose origin and destination are some metaconcept of the concepts.
- Concepts related through undefined instance attributes of some metaconcept of the concepts.
- Instance attributes of an undefined resource, of multiple concepts, or instance attributes not related to a concept or without a type.
- Relations without origin or destination, with multiple origins or destinations, or with undefined resources as origin or destination.
- Instances of undefined or multiple concepts, instances related through undefined relations, or instances with undefined instance attributes.

With regard to the export of concepts and properties whose names include URI character restrictions, WebODE:

- Does not modify the name of a concept, nor of an instance attribute, nor of a relation when it does not start with a letter or ”_”.
- Encodes spaces in concept names, in instance attribute names, and in relation names as ”_”.
- Does not modify the name of a concept, nor of an instance attribute, nor of a relation when it includes URI reserved characters.
- Cannot model concepts and properties with the character ’”’ in their names.

5.4 RDF repositories: Corese, Jena and Sesame

by OLIVIER CORBY, RAÚL GARCÍA-CASTRO AND JESÚS PRIETO-GONZÁLEZ

This section deals with the results of evaluating the RDF(S) export capabilities of Corese, Jena and Sesame, the RDF repositories that participated in the benchmarking.

The procedure for executing the RDF(S) Export Benchmark Suite in the RDF repositories was different from the one proposed in the benchmark suite. This is so because some tasks proposed in the benchmark suite require functionalities available in ontology development tools, such as modelling the ontology into the tool, but not in RDF repositories.

The procedure followed in the RDF repositories was the following:

- To define a fictitious ontology that covered all the combinations of components present in the benchmark suite.
- To define queries for extracting the combinations of components required in each benchmark from the ontology to a RDF file. In the case of Corese, Jena and Sesame the queries were defined in the SPARQL query language[1].
- To export the combinations of components by running the queries against the ontology and saving the results in separate files.
- To compare the exported RDF(S) ontologies with the expected ones, checking whether there is some addition or loss of information.

The RDF repositories export correctly all the combinations of components. This is so because RDF is the knowledge model of these tools, and exporting ontologies to RDF does not require any transformation.

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

Chapter 6

RDF(S) interoperability results and analysis

by RAÚL GARCÍA-CASTRO

This chapter presents the results of executing the RDF(S) Interoperability Benchmark Suite in the ontology development tools participating in the benchmarking. In the first section, a global analysis on the interoperability of the tools is performed, and in the following sections a thorough analysis of each tool is provided.

As the ontology development tools participating in the benchmarking have different knowledge models, both the experimentation and the analysis of the results are based on a common group of ontology components that is present in these tools. Therefore, the knowledge models of the tools participating in the benchmarking cover more or less this common group.

The import and export results presented in the previous chapters showed few problems when importing and exporting ontologies but in this chapter we present quite a few interoperability problems.

As a general comment, interoperability between the tools depends on:

a. The correct working of their RDF(S) importers and exporters.

b. The way chosen for serializing the exported ontologies in the RDF/XML syntax.

Furthermore, we have observed that some problems in any of these factors affect the results of not just one but several benchmarks. This means that in some cases correcting a single import or export problem or changing the way of serializing ontologies can cause significant interoperability improvements.

Next, we list the components that can be interchanged between the tools. These components are summarized in Table 6.1, where each column shows if the combination of components can be interchanged between a group of tools or not[1]. The ”-” character means that the component cannot be modelled in some of the tools and therefore cannot be interchanged between them.

Interoperability using the same tool

When the source and the destination of an ontology interchange are the same tool, no problem occurs in this interchange. The only exception resides in Protégé, as shown below.

When **KAON** interoperates with itself, it interchanges correctly all the common components that it can model.

When **Protégé** interoperates with itself, it also interchanges correctly almost all the common components that it can model. The exception occurs when Protégé interchanges classes that are instances of multiple metaclasses and instances of multiple classes, because it does not import resources that are instances of multiple metaclasses.

When **WebODE** interoperates with itself, it interchanges correctly all the common components that it can model.

Interoperability between each pair of tools

Interoperability between different tools varies depending on the tools. Besides, as the detailed interoperability results show, in some cases the tools are able to interchange certain components from one tool to another, but not the other way round.

When **KAON** interoperates with **Protégé**, they can interchange correctly some of the common components that these tools are able to model. But problems occur with classes that are instance of a single metaclass or of multiple metaclasses, with datatype properties without domain or range, with datatype properties whose range is *String*, with instances of multiple classes, and with instances related through datatype properties.

When **KAON** interoperates with **WebODE**, they can interchange correctly almost all the common components that these tools can model. The only exception occurs when they interchange datatype properties with domain and whose range is *String*.

When **Protégé** interoperates with **WebODE**, they can interchange correctly all the common components that these tools can model.

[1]The names of the tools have been shortened in the heading of the table: KAON=K, Protégé=P and WebODE=W

Combination of components	**K-K**	**P-P**	**W-W**	**K-P**	**K-W**	**P-W**	**K-P-W**
Classes	Y	Y	Y	Y	Y	Y	Y
Classes instance of a single metaclass	Y	Y	-	N	-	-	-
Classes instance of a multiple metaclasses	Y	N	-	N	-	-	-
Class hierarchies without cycles	Y	Y	Y	Y	Y	Y	Y
Datatype properties without domain or range	Y	Y	-	N	-	-	-
Datatype properties with multiple domains	Y	-	-	-	-	-	-
Datatype properties whose range is String	Y	Y	Y	N	N	Y	N
Datatype properties whose range is a XML Schema datatype	Y	-	Y	-	Y	-	-
Object properties without domain or range	Y	Y	-	Y	-	-	-
Object properties with multiple domains or ranges	Y	-	-	-	-	-	-
Object properties with a domain and range	Y	Y	Y	Y	Y	Y	Y
Instances of a single class	Y	Y	Y	Y	Y	Y	Y
Instances of multiple classes	Y	N	-	N	-	-	-
Instances related through object properties	Y	Y	Y	Y	Y	Y	Y
Instances related through datatype properties	Y	Y	Y	N	Y	Y	N
Instances related through datatype properties whose range is a XML Schema datatype	-	-	Y	-	-	-	-

Table 6.1: Components interchanged between the tools

Interoperability between all the tools

Interoperability between **KAON**, **Protégé** and **WebODE** can be achieved by means of nearly all the common components that these tools can model. The only common components that these tools cannot use are datatype properties with domain and whose range is *String* and instances related through datatype properties.

Interoperability regarding URI character restrictions

Interoperability is low when tools interchange ontologies containing URI character restrictions in class and property names. This is mainly due to the fact that tools usually encode some or all the characters that do not comply with these restrictions, which provokes changes in class and property names.

KAON can interchange with itself ontologies having URI character restrictions in class and property names.

Protégé can interchange neither with itself nor with **KAON** nor with **WebODE** ontologies having URI character restrictions in class and property names.

WebODE can interchange with itself ontologies having class and property names that do not start with a letter or "_" and with spaces in their names.

KAON and **WebODE** can only interchange ontologies having class and property names that do not start with a letter or "_".

The next sections present a summary of the analysis carried out on each tool. A detailed analysis of the RDF(S) interoperability results can be found in Appendix C.

6.1 KAON results

by YORK SURE AND MARKUS ZONDLER

This section includes the results of evaluating the RDF(S) interoperability capabilities of KAON.

Table 6.2 shows the combinations of components that can be modelled in KAON and explains whether these combinations can be interchanged from the tools to KAON or not. The cells in this table (and in the rest of the tables of this chapter) include:

- ***OK*** when the *Execution* results of all the benchmarks where the combination of components appears is *OK*.
- ***FAIL*** when the *Execution* result of some benchmark where the combination of

components appears is *FAIL*.

- **'-'** when the combination of components cannot be modelled in the source tool.

Combination of components	**Corese**	**KAON**	**Protégé**	**WebODE**
Classes	OK	OK	OK	OK
Classes instance of a single metaclass	OK	OK	OK	-
Classes instance of a multiple metaclasses	OK	OK	FAIL	-
Class hierarchies without cycles	OK	OK	OK	OK
Datatype properties without domain and range	OK	OK	OK	-
Datatype properties with domain but without range	OK	OK	OK	-
Datatype properties without domain but with range	OK	OK	FAIL	-
Datatype properties with multiple domains	OK	OK	-	-
Datatype properties whose range is String	OK	OK	FAIL	FAIL
Datatype properties with domain and whose range is a XML Schema datatype	OK	OK	-	OK
Object properties without domain and range	OK	OK	OK	-
Object properties with domain but without range	OK	OK	OK	-
Object properties without domain but with range	OK	OK	OK	-
Object properties with multiple domains	OK	OK	-	-
Object properties with multiple ranges	OK	OK	-	-
Object properties with a domain and range	OK	OK	OK	OK
Instances of a single class	OK	OK	OK	OK
Instances of multiple classes	OK	OK	OK	-
Instances related through object properties	OK	OK	OK	OK
Instances related through datatype properties	FAIL	OK	FAIL	OK

Table 6.2: Components modelled in KAON interchanged correctly

In Table 6.2 we can see the combinations of components that can be interchanged among the tools that can model the combination of components and KAON. These combinations are:

- **Classes, classes that are instances of a single metaclass, and class hierarchies without cycles**.

- **Datatype properties without domain and range, with domain and without range, or with multiple domains**.
- **Object properties with a domain and a range, without domain or range, or with multiple domains or ranges**.
- **Instances of a single class or of multiple classes, and instances related through object properties**.

The other combinations of components are not interchanged, though KAON can model them, for the following reasons:

- **Classes that are instances of multiple metaclasses.** KAON cannot receive classes that are instances of multiple metaclasses from Protégé.
- **Datatype properties without domain and with range.** KAON cannot receive datatype properties without domain and with range from Protégé because the information about the range is lost.
- **Datatype properties whose range is *String*.** KAON cannot receive datatype properties whose range is *String* from Protégé and WebODE because the information about the range is lost.
- **Instances related through datatype properties.** KAON cannot receive instances related through datatype properties from Protégé because the information about the range is lost.

The combinations of components of the common knowledge model of the tools that cannot be modelled and, therefore, cannot be interchanged with KAON are the following:

- **Class hierarchies with cycles**.
- **Classes related through object or datatype properties**.
- **Object and datatype properties with undefined resources as domain or range.**
- **Instances of undefined resources or related through undefined object and datatype properties.**

With regard to the interchange of classes and properties with URI character restrictions in their names, KAON:

- Interchanges with WebODE and itself classes and properties whose name starts with a character that is not a letter nor '_' , but it does not interchange them with Protégé because the information about the range of the property is not imported.

- Interchanges with WebODE and itself classes and properties with spaces in their names but it does not interchange them with Protégé because the information about the range of the property is not imported.
- Interchanges with WebODE and itself classes and properties with URI reserved characters in their names, but it does not interchange them with Protégé because the information about the range of the property is not imported.
- Interchanges with itself classes and properties with XML delimiter characters in their names, but it does not interchange them with Protégé because the information about the range of the property is not imported.

6.2 Protégé results

by RAÚL GARCÍA-CASTRO

This section presents the results of evaluating the RDF(S) interoperability capabilities of Protégé.

Table 6.3 shows the combinations of components that can be modelled in Protégé and tells whether these combinations can be interchanged from the tools to Protégé or not.

In Table 6.3 we can see the combinations of components can be interchanged between the tools that can model the combination of components and Protégé. These combinations are:

- **Classes and class hierarchies without cycles**.
- **Object properties with a domain and range or without domain or range**. In the last case, the object property is created with a range of *Any*.
- **Instances of a single class**.

The other combinations of components are not interchanged, although Protégé can model them, for the following reasons:

- **Classes that are instances of a single metaclass.** Protégé does not import that classes are instances of multiple metaclasses (even if one of them is *rdfs:Class*). Therefore, Protégé cannot import that classes are instances of a single metaclass if in the file they also appear as instances of *rdfs:Class*. We propose two solutions for this problem:
  - When Corese and KAON export classes that are instances of a metaclass, they should export the classes as instances of just the metaclass and not of *rdfs:Class*.

Combination of components	**Corese**	**KAON**	**Protégé**	**WebODE**
Classes	OK	OK	OK	OK
Classes instance of a single metaclass	FAIL	FAIL	OK	-
Classes instance of a multiple metaclasses	FAIL	FAIL	FAIL	-
Class hierarchies without cycles	OK	OK	OK	OK
Datatype properties without domain and range	FAIL	FAIL	OK	-
Datatype properties with domain but without range	OK	FAIL	OK	-
Datatype properties without domain but with range	FAIL	OK	OK	-
Datatype properties whose range is String	FAIL	OK	OK	OK
Object properties without domain and range	OK	OK	OK	-
Object properties with domain but without range	OK	OK	OK	-
Object properties without domain but with range	OK	OK	OK	-
Object properties with a domain and range	OK	OK	OK	OK
Instances of a single class	OK	OK	OK	OK
Instances of multiple classes	FAIL	FAIL	FAIL	-
Instances related through object properties	FAIL	OK	OK	OK
Instances related through datatype properties	FAIL	OK	OK	OK

Table 6.3: Components modelled in Protégé interchanged correctly

  – When Protégé imports a class that is instance of multiple metaclasses (including *rdfs:Class*), it should import it correctly.

- **Classes instance of multiple metaclasses.** Protégé does not import that classes are instances of multiple metaclasses (even if one of them is *rdfs:Class*). Therefore, Protégé cannot import that classes are instances of multiple metaclasses. We propose one solution for this problem:

  – When Protégé imports a class that is instance of multiple metaclasses (including *rdfs:Class*), it should import it correctly.

- **Datatype properties without domain or range, or with range *String*.** Protégé crashes and does not import anything when a XML Schema datatype (i.e. *xsd:integer*) is defined in the file as a *rdfs:Datatype*. We propose two solutions for this problem:

  – When Corese exports XML Schema datatypes, it should not export them as *rdfs:Datatype*.

  - When Protégé imports a XML Schema datatype defined as *rdfs:Datatype*, it should not crash.

- **Instances of multiple classes.** Protégé does not import instances that are instances of multiple classes. We propose one solution for this problem:

  - When Protégé imports an instance that is instance of multiple classes, it should import it correctly.

- **Instances related through object and datatype properties.** Protégé crashes when it imports properties with *rdf:datatype* attributes. We propose two solutions for this problem:

  - When Corese exports instances related through properties, it should not export *rdf:datatype* attributes in the properties.
  - When Protégé imports a *rdf:datatype* attribute in a property, it should not crash.

The combinations of components of the common knowledge model of the tools that cannot be modelled and, therefore, cannot be interchanged with Protégé are the following:

- **Class hierarchies with cycles**. When Protégé finds a cycle in a class hierarchy from Corese, it crashes and does not import anything. We propose one solution for this problem:

  - When Protégé imports class hierarchies with cycles, it should not crash.

- **Classes related through object properties**. Protégé does not import the property between the class and another class nor the domain and range of the property.

- **Classes related through datatype properties**. Protégé does not import the property between the class and a datatype nor the domain and range of the property.

- **Object and datatype properties with undefined resources as domain or range.** Protégé creates the undefined resource as a class.

- **Object and datatype properties with multiple domains.** Protégé creates the property as a template slot with multiple domains. This is not the expected result because in Protégé multiple domains in slots are considered as the union of all the domains and in RDF(S) multiple domains in properties are considered the intersection of all the domains. We propose one solution for this problem:

  - When Protégé imports object and datatype properties with multiple domains, it should import them as template slots with no domain, as it occurs when it imports object properties with multiple ranges.

- **Object properties with multiple ranges.** Protégé creates the property as a template slot with a range of *Any*.
- **Datatype properties whose range is a XML Schema datatype.** When receiving ontologies from Corese, Protégé crashes and does not import anything when a XML Schema datatype (i.e. *xsd:integer*) is defined in the file as a *rdfs:Datatype*. We propose two solutions for this problem:
  - When Corese exports XML Schema datatypes, it should not export them as *rdfs:Datatype*.
  - When Protégé imports a XML Schema datatype defined as *rdfs:Datatype*, it should not crash.

  When receiving from KAON and WebODE, Protégé creates the XML Schema datatype as a class, which is the range of the property.
- **Instances of undefined resources.** Protégé creates the undefined resource as a class.
- **Instances related through undefined object and datatype properties.** Protégé creates the property as a template slot without a domain and with a range of *Any*. The property between the instances is not created.
- **Instances related through datatype properties whose range is a XML Schema datatype.** Protégé creates the XML Schema datatype as a class and the range of the property is this class. But it does not import the literal value of the property. We propose one solution for this problem:
  - When Protégé imports instances related through datatype properties whose range is a XML Schema datatype, it should import the literal value as an instance of the XML Schema datatype class.

Regarding the interchange of classes and properties with URI character restrictions in their names, Protégé:

- Interchanges with KAON and WebODE classes and properties whose name start with a character that is not a letter nor '_', but it does not interchange them with itself because Protégé replaces when exporting the illegal character with '_'.
- Does not interchange with any tool classes and properties having spaces in their names because all the tools replace when exporting the illegal character with '_'.
- Interchanges with WebODE classes and properties with URI reserved characters in their names, but it does not interchange them with KAON and itself because they replace when exporting the illegal character with '_'.

- Does not interchange with any tool classes and properties having XML delimiter characters in their names.

6.3 WebODE results

by RAÚL GARCÍA-CASTRO

This section describes the results of evaluating the RDF(S) interoperability capabilities of WebODE.

Table 6.4 shows the combinations of components that can be modelled in WebODE and whether these combinations can be interchanged from the tools to WebODE or not.

Combination of components	**Corese**	**KAON**	**Protégé**	**WebODE**
Classes	OK	OK	OK	OK
Class hierarchies without cycles	FAIL	OK	OK	OK
Datatype properties with domain and whose range is *String*	FAIL	OK	OK	OK
Datatype properties with domain and whose range is a XML Schema datatype	OK	OK	-	OK
Object properties with a domain and a range	OK	OK	OK	OK
Instances of a single class	OK	OK	OK	OK
Instances related through object properties	OK	OK	OK	OK
Instances related through datatype properties whose range is String	OK	OK	OK	OK
Instances related through datatype properties whose range is a XML Schema datatype	OK	-	-	OK

Table 6.4: Components modelled in WebODE interchanged correctly

In Table 6.4 we can see the combinations of components that can be interchanged among the tools that can model the combination of components and WebODE. These are:

- **Classes**.
- **Datatype properties with a domain and whose range is a XML Schema datatype**.
- **Object properties with a domain and a range**.
- **Instances of a single class**.
- **Instances related through object properties, or through datatype properties whose range is *String* or a XML Schema datatype**.

The remainder combinations of components are not interchanged even though WebODE can model them. The reasons for this are:

- **Class hierarchies without cycles**. WebODE does not import the subclass properties if the superclass is not defined as a class in the file. We propose two solutions for this problem:
  - When Corese exports class hierarchies without cycles, it should export the superclasses as classes in the file.
  - When WebODE imports superclasses, it should import them as classes, even if they are not defined in the file.
- **Datatype properties with domain and whose range is *String*.** WebODE crashes when the *String* range is defined in the file as a datatype without namespace. We propose two solutions for this problem:
  - When Corese exports the range *String*, it should export it with a namespace.
  - When WebODE imports files with datatypes without namespace, it should not crash.

The combinations of components that cannot be modelled and, therefore, cannot be interchanged with WebODE are the following:

- **Classes that are instances of metaclasses.**

  WebODE cannot model metaclasses. Therefore, when it receives metaclasses from other tools, it imports the metaclasses as classes and loses the *rdf:type* properties between classes. If a metaclass is not defined as a class in the exported file, the metaclass is not imported. We propose two solutions for this problem:
  - When Corese and Protégé export a metaclass, they should define it as a class in the file.
  - When WebODE imports a metaclass, it should import it correctly even if it is not defined as a class in the file.

  If the class is not defined as a class in the exported file, the class is imported as an instance. We propose one solution for this problem:
  - When Protégé exports a class instance of a metaclass, to define the class as a class in the file.
- **Class hierarchies with cycles**. When WebODE finds a cycle in a class hierarchy from Corese, it creates a class and an imported term with the same name as the object of the *rdfs:subClassOf* property that causes the cycle and creates the subclass with the imported term. We propose one solution for this problem:

  - When WebODE imports a hierarchy with cycles, it should create the hierarchy removing the *rdfs:subClassOf* properties that form the cycles.

- **Classes related through object properties**. WebODE does not import the property.

- **Classes related through datatype properties**. WebODE does not import the property.

- **Object and datatype properties without domain and without range**. When WebODE imports an object or a datatype property without domain and range, it creates *rdfs:Resource* as an imported term and the property as an object property with a domain and range of *rdfs:Resource*.

- **Object and datatype properties with domain and without range**. When WebODE imports an object or a datatype property with domain but without range, it creates *rdfs:Resource* as an imported term and the property as an object property with a range of *rdfs:Resource*.

- **Object and datatype properties without domain and with range**.

  When WebODE imports an object or a datatype property without domain but with range, it creates *rdfs:Resource* as an imported term and the property with a domain of *rdfs:Resource*.

- **Object and datatype properties with undefined resources as domain or range.** WebODE creates the undefined resource as a class.

- **Object and datatype properties with multiple domains.** WebODE imports an object or datatype property that has multiple domains, creating an anonymous concept as the domain of the datatype property and as subclass of one of the domain classes. One proposed solution for this problem is:

  - When WebODE imports an object or datatype property with multiple domains, it should create the anonymous concept as subclass of all the domain classes.

- **Object properties with multiple ranges.** WebODE imports an object property that has multiple ranges by creating an anonymous concept as the range of the property and as subclass of one of the range classes. One proposed solution for this problem is:

  - When WebODE imports an object property with multiple ranges, it should create the anonymous concept as subclass of all the range classes.

- **Instances of undefined resources.** WebODE does not import the instance.

- **Instances of multiple classes.** WebODE imports the instance as instance of just one class.
- **Instances related through undefined object and datatype properties.** WebODE does not import the undefined properties.

Regarding the interchange of classes and properties with URI character restrictions in their names, WebODE:

- Interchanges with KAON and itself classes and properties whose name start with a character that is not a letter nor '_', but does not interchange them with Protégé because Protégé replaces when exporting the illegal character with '_'.
- Does not interchange with any tool classes and properties with spaces in their names, as the tools replace when exporting the illegal character with '_'.
- Interchanges with itself classes and properties with URI reserved characters in their names, but it does not interchange them with KAON and Protégé because these tools replace when exporting the illegal character with '_'.
- Does not interchange classes and properties having XML delimiter characters in their names with any tool.

Chapter 7

RDF(S) interoperability in the use cases

by ELENA PASLARU BONTAS, LYNDON NIXON, MALGORZATA MOCHOL, AND RAÚL GARCÍA CASTRO

This chapter focuses on the utilization of the RDF(S) interoperability benchmarking in the context of the business use cases collected from Knowledge Web Industry Board partners and published in the deliverable D1.1.2 (Prototypical Business Use Cases) [Nixon *et al.*, 2004]. Following the establishment of a typology of knowledge processing components/tasks, which are common to real world business scenarios, in D1.1.3 (Knowledge processing tasks) [Shvaiko *et al.*, 2004] these use cases have been further analyzed in order to extract core research challenges in these terms, which can be delivered to Knowledge Web researchers, in deliverable D1.1.4 (System and knowledge technology components for prototypical applications and business cases) [Leger *et al.*, 2005].

Starting with the latter, we concentrate here on those use cases in which the interoperability issue has been identified as a research question which needs further investigation and which could take benefit from the results achieved in work packages 1.2 and 2.2.

In the use cases, the need of interoperability mainly appears at two levels (not always separated). One is during the development and maintenance of the ontologies, where a group of ontology developers must jointly develop and interchange ontologies. And the other is during the use of the ontologies, where some tools need to interchange ontologies for their correct working.

The collaborative ontology engineering process is inconceivable without interoperable ontology management tools. Local ontologies are developed independently possibly importig external ontologies or using different supporting tools and different engineering approaches (from scratch, using ontology learning techniques etc.). Further on, the knowledge modelled with these ontologies might be stored in various semantic repositories.

Next, we describe the use cases where interoperability is critical to their success and research solutions must be provided to accomplish them. In the other use cases interoperability, although relevant, is not critical to accomplish them, being a secondary research problem.

Every use case will be described in the following way: first we will give a brief overview of the business scenario outlining the need for semantic technologies, then we will introduce the identified RDF(S) interoperability requirements of the scenario.

7.1 Use Case 1. Recruitment from Worldwidejobs

Topic: eRecruitment

Knowledge Web Partner: Free University Berlin

Industry Board Member: World Wide Jobs GmbH

The use case focuses on the usage of semantic technologies in the eRecruitment domain for the realization of an ontology-based matching and ranking engine for job portals.

Requirements for RDF(S) interoperability

The semantic job portal will have to store huge amounts of collected RDF data and to provide high performance access to this data for its users. This raises the problem of scalability and efficient retrieval in current RDF storage solutions, but also the question of interoperability between the RDF(S) repositories and other non-RDF legacy data sources. In the current situation, if interoperability is not achieved with information providers that manage non-RDF data, the information coverage of the semantic portal will be very limited.

A second aspect of the eRecruitment scenario is related to the creation and maintenance of the ontological structures underlying the semantic portal implementation. As eRecruitment ontologies are subject to frequent changes in loosely coupled development teams, there is a need for interoperable ontology management environments, in order to allow a dispersed community of ontology developers to interact with the eRecruitment ontologies in a comfortable and in the same time efficient manner.

7.2 Use Case 3. News Aggregation from Neofonie

Topic: News aggregation service

Knowledge Web Partner: Free University Berlin

Industry Board Member: Neofonie GmbH

The business case deals with the realization of an aggregated news service which provides business clients with advanced information management capabilities. Semantic technologies can be applied for the implementation of many of these services, from the classification and annotation of the news items using ontologies to content-based retrieval.

Requirements for RDF(S) interoperability

One of the difficulties of this use case is that of translating news from heterogeneous information sources storing their data in different formats and with different underlying representation formalisms. As in the previous case, the news coverage depends on the ability of obtaining news from as much sources as possible.

Another research challenge of this use case is the development and maintenance of the ontology/ontologies modelling the domain of the aggregated news collections. These tasks take benefit from the availability of interoperable technologies, which do not constrain the engineering team in their choices upon a particular technological environment for developing and maintaining ontologies.

7.3 Use Case 4. Product Lifecycle Management from Semtation

Topic: Product lifecycle management

Knowledge Web Partner: Free University Berlin

Industry Board Member: Semtation GmbH

Product lifecycle management is currently based around the manual development of a product catalogue in which the product knowledge and relationships are defined by the developers at the implementation stage. Using semantic technologies such catalogues could be refined by means of formal ontologies describing products and their core properties. The usage of ontologies in this context forms a basis for various information services related to the product descriptions, such as ontology-based search and product comparisons.

Requirements for RDF(S) interoperability

In order to tap the full potential of ontologies as means to model product-related information there is a need for tools supporting domain experts in the knowledge explicitation task. We can identify that there are different levels of ontology that will need to be created, possibly by different management tasks. The tool developer will need an ontology that models the common aspects of product catalogues. The tool user will need an ontology (or ontologies) that can express knowledge about his or her type of product. Finally, users of the product catalogue data (not of the tool) may have their own requirements that will have to be met.

Each of these types of users have different expertise with the ontology development process and model different aspects of the ontologized product catalogue. Therefore, they require different tools for developing the ontology and the interoperability of these tools in this setting is essential for the accomplishment of this task.

Chapter 8

Recommendations

by RAÚL GARCÍA-CASTRO

8.1 Recommendations for ontology developers

This section offers recommendations for ontology developers when using ontology development tools to build ontologies with the goal of interchanging them with other tools.

If the ontology is being developed bearing in mind interoperability, ontology developers should be aware of the components that can be represented in the ontology development tools and in RDF(S). Furthermore, they should try to use the common components of these tools in their ontologies to avoid the already known knowledge losses.

Ontology developers should also be aware of the semantic equivalences and differences between the knowledge models of the tools and the interchange language. For example, in Protégé multiple domains in template slots are considered the union of all the domains while in RDF(S) multiple domains in properties are considered the intersection of all the domains; in WebODE instance attributes are local to a single concept while in RDF(S) properties are global and can be used in any class.

Depending on the tools that participate in the ontology interchange, the level of interoperability is greater or lower, as can be seen in Chapter 6. The benchmarking results show that interoperability can be achieved among the tools that have participated in the benchmarking using:

- Classes.
- Class hierarchies without cycles.
- Object properties with domain and with range.
- Instances of a single class.

- Instances related through object properties.

Also, it is not recommended to name resources including in their names spaces or any character that is restricted in the RDF(S), URI or XML specifications.

In the case of interoperability in the RDF repositories, although these repositories export and import correctly to RDF(S), users should consider the limitations that other tools have when exporting their ontologies to RDF(S) with the aim of interchanging them.

8.2 Recommendations for software developers

This section includes general recommendations for improving the interoperability of the tools when developing them as well as some guidelines for developers to enhance each of the participating tools according to the results and to the practices found.

Interoperability between ontology development tools using RDF(S) as interchange language depends on how the importers and exporters of these tools work. And how these importers and exporters work depends on the development decisions made by tool developers, who are different people with different needs. Therefore, it is not straightforward to provide general recommendations for developers. Nevertheless, some comments can be extracted from the analysis of the benchmarking results:

- In a few cases, a development decision will produce an interoperability improvement with some tools but a loss with others. For example, when exporting classes that are instances of a metaclass, some tools require that the class be defined as instance of *rdfs:Class* while some other tools require the opposite.

- The collateral consequences of the development decisions should be analysed by the tool developers. For example, if a datatype is imported as a class in the ontology, then the literal values of this datatype should be imported as instances in the ontology, which would complicate the management of these values.

- Tool developers and ontology developers should be aware of the semantic equivalences and differences between the knowledge models of their tool and the interchange language; on the other hand, the tools should notify the user when the semantics is changed.

- The first requirement for achieving interoperability is that the importers and exporters of the tools are robust and work correctly when dealing with unexpected inputs. Although this is an evident comment, the results show that this requirement is not fulfilled by the tools and even some tools crash when importing some combinations of components.

- Above all, tools should deal correctly with the combinations of components that can be present in the interchange language but cannot be modelled in them. For example, cycles in class and property hierarchies cannot be modelled in ontology development tools. Nevertheless, these tools should be able to import these hierarchies eliminating the cycles.
- When exporting components commonly used by ontology development tools, they should be completely defined in the file. This means that metaclasses and classes in class hierarchies should be defined as instances of *rdfs:Class*, properties should be defined as instances of *rdf:Property*, etc.
- Exporting complete definitions of other components can cause problems if these are imported by other tools. Not every tool deals with datatypes defined as instances of *rdfs:Datatype* in the file or with *rdf:datatype* attributes in properties.
- Every exported resource should have a namespace if the document does not define a default namespace.

Next, we offer some recommendations to improve each of the participating tools according to the results and to the practices found. Although it is not compulsory to follow these recommendations, they help correct interoperability problems as it was detected when analysing the results.

For **Corese** to improve its interoperability with the other tools participating in the benchmarking, it should:

- Export metaclasses defining them as classes in the file.
- Export class hierarchies defining all the superclasses as classes in the file.
- Not export classes that are instances of a metaclass as classes that are instances of *rdfs:Class*.
- Not export datatypes without a namespace.
- Not export XML Schema datatypes as a *rdfs:Datatype* in the file.
- Not include *rdf:datatype* attributes in properties when exporting instances related through properties.

For **KAON** to improve its interoperability with the other tools participating in the benchmarking, it should:

- Not export classes that are instances of a metaclass as classes that are instances of *rdfs:Class*.

- Not export datatype properties without range as datatype properties with a range of *rdfs:Literal*.
- Export all the domains when exporting datatype properties with multiple domains and a XML Schema datatype as range.
- Not crash when importing class hierarchies with cycles.

In order for **Protégé** to improve its interoperability with the other tools participating in the benchmarking, it should:

- Export correctly classes that are instances of multiple metaclasses.
- Import correctly classes that are instances of multiple metaclasses (including *rdfs:Class*).
- Export correctly instances that are instances of multiple classes.
- Import correctly instances that are instances of multiple classes.
- Export classes that are instances of a metaclass defining them as a class in the file.
- Export metaclasses defining them as a class in the file.
- Not crash when importing XML Schema datatypes defined as a *rdfs:Datatype*.
- Not crash when importing *rdf:datatype* attributes in properties.
- Not crash when importing class hierarchies with cycles.
- Not crash when importing property hierarchies with cycles.
- Export object and datatype properties with multiple domains as properties with no domain, as it occurs in the case of exporting object properties with multiple ranges.
- Import object and datatype properties with multiple domains as template slots with no domain, as it occurs when importing object properties with multiple ranges.
- Import the literal value as an instance of the XML Schema datatype class when importing instances related through datatype properties whose range is a XML Schema datatype.
- Not crash when importing XML Schema datatypes defined as a *rdfs:Datatype*.
- Not crash when importing *rdf:datatype* attributes in properties.
- Not crash when importing class hierarchies with cycles.
- Not crash when importing property hierarchies with cycles.

- Not crash when importing resources with empty nodes in the unshortened syntax.
- Not crash when importing resources with blank nodes in the unshortened syntax.

For **WebODE** to improve its interoperability with the other tools participating in the benchmarking, it should:

- Import superclasses in class hierarchies as classes, even if they are not defined in the file.
- Not crash when importing files with datatypes without namespace.
- Import metaclasses even if they are not defined as classes in the file.
- Remove *rdfs:subClassOf* properties that form cycles when importing class hierarchies with cycles.
- Create the anonymous concept as subclass of all the domain classes when importing datatype and object properties with multiple domains.
- Create the anonymous concept as subclass of all the range classes when importing object properties with multiple ranges.

8.3 Feasibility of RDF(S) interoperability in the use cases

This section includes an analysis of the feasibility of the industry use cases interoperability requirements stated in Chapter 7, when dealing with ontology interchange using RDF(S) as interchange language.

The use cases considered in Knowledge Web show the need of developing and maintaining ontologies present in heterogeneous Semantic Web resources and created using different approaches (from scratch, using ontology learning techniques, etc.). Also, ontology repositories are one of the preferred options for ontology storage and share. Therefore, ontology development tools should be able of interoperating with these repositories and resources.

Benchmarking results show us that there is no problem with interoperability when only using native RDF(S) tools.

The problem is that the profile of ontology developers is also highly heterogeneous. The developers of an ontology are usually geographically dispersed, belong to different organisations, have different levels of expertise with the technology, and use different tools in the ontology development process. Therefore, using RDF(S) native tools may not be the correct approach in some of the use cases.

In any case, developers and users should be aware of the side effects of interchanging ontologies, as interoperability using ontology development tools with different knowledge models is limited to a subset of ontology components depending on the tools that interchange the ontology.

In order to transact the use cases some decisions will have to be made for solving interoperability issues. This deliverable provides guidelines for helping in making these decisions and contains the interoperability results of the tools that took part of the benchmarking.

8.4 Recommendations for benchmarking

This section offers recommendations to perform benchmarking activities, extracted from the lessons learnt while instantiating the methodology.

Benchmarking needs the participation of relevant experts in the domain together with the best-in-class tools.

Resources are needed mainly in three tasks: benchmarking organisation, definition of the experimentation, and execution of the experiments and analysis of the results. It should be ensured that enough resources are allocated to each of these tasks.

Benchmarking is not about comparing the results of the tools but the practices that lead to these results. Therefore, experimentation should be designed to obtain these practices as well as the results.

In this benchmarking activity, the experimentation over the tools was done almost manually. Carrying out manual experiments and analysing the results is expensive. Therefore, these tasks should be automated as much as possible.

Another drawback of performing experiments manually is that it can be influenced by human mistakes. To avoid this, some mechanisms should be set up to detect this kind of errors.

Benchmarking is an activity that is lasting as it requires tasks that are not immediate: announcements, agreements, etc. Therefore, benchmarking activities should start early in time and the benchmarking planning should consider a realistic duration of the benchmarking.

Chapter 9

Conclusion

by RAÚL GARCÍA-CASTRO

This document is intended to serve not just as a summary of the RDF(S) interoperability benchmarking, but as a guide for people who want to perform benchmarking activities or interoperability evaluations over Semantic Web technology.

The results of the benchmarking can also be used by ontology development tool users that have problems when interchanging ontologies or want to foresee the results of a future interchange.

The benchmark suites that have been used in the benchmarking can be used by any tool with RDF(S) import and export capabilities, with no need to participate in the benchmarking, and can be useful both while the tool is being developed and after its development has finished.

Although it is not required that the developers of the tool participate in the benchmarking and perform the experiments over their tool, their involvement facilitates the execution and analysis of the experimentation results to a large extent.

In all the cases where tool developers performed the experimentation over their own tools, tool improvement has occurred before the *Improve* phase of the methodology, as developers were able to detect problems and correct their tools while executing the benchmark suites.

The manual execution of the experiments and analysis of the results causes the benchmark suite to be costly. Sometimes tool developers have automated the execution of the benchmark suites, but not always. On the other hand, work has started to give some means of analysing the experimentation results automatically and to provide functionalities for updating the experimentation results or including the results of new tools.

In order to automate both the execution of the benchmark suites and the analysis of the results, having machine-readable descriptions of the benchmarks (i.e. in RDF) would be very useful.

Another drawback of the manual execution of experiments is that the results obtained depend on the people performing these experiments, on their expertise with the tools and on their ability to extract the practices performed for developing the tools.

Bibliography

[Arpírez *et al.*, 2003] J.C. Arpírez, O. Corcho, M. Fernández-López, and A. Gómez-Pérez. WebODE in a nutshell. *AI Magazine*, 24(3):37–47, Fall 2003.

[Brickley and Guha, 2004] D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation 10 February 2004, 2004.

[D1.2.2.1.2, 2007] D1.2.2.1.2 Benchmarking the interoperability of ontology development tools using OWL as interchange language. Technical report, Knowledge Web, To appear in June 2007.

[García-Castro and Gómez-Pérez, 2005a] R. García-Castro and A. Gómez-Pérez. A method for performing an exhaustive evaluation of RDF(S) importers. In *Proceedings of the Workshop on Scalable Semantic Web Knowledge Based Systems (SSWS2005)*, number 3807 in LNCS, pages 199–206, New York, USA, November 2005. Springer-Verlag.

[García-Castro and Gómez-Pérez, 2005b] R. García-Castro and A. Gómez-Pérez. A method for performing an exhaustive evaluation of RDF(S) importers. In *Proceedings of the Workshop on Scalable Semantic Web Knowledge Based Systems (SSWS2005)*, number 3807 in LNCS, pages 199–206, New York, USA, November 2005. Springer-Verlag.

[García-Castro and Gómez-Pérez, 2006a] R. García-Castro and A. Gómez-Pérez. Benchmark suites for improving the rdf(s) importers and exporters of ontology development tools. In *Proceedings of the 3rd European Semantic Web Conference (ESWC2006)*, LNCS 4011, Budva, Montenegro, June 2006. Springer-Verlag.

[García-Castro and Gómez-Pérez, 2006b] R. García-Castro and A. Gómez-Pérez. Interoperability of protégé using RDF(S) as interchange language. In *Proceedings of the 9th International Protégé Conference (Protégé2006)*, Stanford, USA, July 2006.

[García-Castro *et al.*, 2004] R. García-Castro, D. Maynard, H. Wache, D. Foxvog, and R. González-Cabero. D2.1.4 specification of a methodology, general criteria and benchmark suites for benchmarking ontology tools. Technical report, Knowledge Web, December 2004.

[García-Castro, 2005] R. García-Castro. D2.1.5 prototypes of tools and benchmark suites for benchmarking ontology building tools. Technical report, Knowledge Web, December 2005.

[Leger *et al.*, 2005] A. Leger, L. Nixon, M. Mochol, F. Paulus, L. Rocuet, M. Bonifacio, R. Cuel, M. Jarrar, Y. Kompatsiaris, V. Papastathis, and S. Dasiopoulou. D1.1.4 v1 system and knowledge technology components for prototypical applications and business cases. Technical report, Knowledge Web, June 2005.

[Nixon *et al.*, 2004] L. Nixon, M. Mochol, A. Leger, F. Paulus, L. Rocuet, M. Bonifacio, R. Cuel, M. Jarrar, P. Verheyden, Y. Kompatsiaris, V. Papastathis, S. Dasiopoulou, and A. Gómez-Pérez. D1.1.2 prototypical business use cases. Technical report, Knowledge Web, December 2004.

[Shvaiko *et al.*, 2004] P. Shvaiko, L. Nixon, M. Mochol, A. Leger, F. Paulus, L. Rocuet, Y. Kompatsiaris, V. Papastathis, and S. Dasiopoulou. D1.1.3 knowledge processing requirements analysis. Technical report, Knowledge Web, December 2004.

Acknowledgments

Thanks to all the people that have participated in the RDF(S) interoperability benchmarking by the time of writing this deliverable: Olivier Corby, York Sure, Moritz Weiten, and Markus Zondler. Without their effort, this could have not been possible.

Thanks to Rosario Plaza for reviewing the grammar of this deliverable.

Related deliverables

A number of Knowledge web deliverables are clearly related to this one:

Project	Number	**Title** and relationship
KW	D1.1.2	**Prototypical business use cases** provided a description of the use cases that are being considered in Knowledge Web.
KW	D2.1.4	**Specification of a methodology, general criteria and benchmark suites for benchmarking ontology tools** presented the benchmarking methodology that has been used for benchmarking the interoperability of ontology development tools using RDF(S) as interchange language.
KW	D2.1.5	**Prototypes of tools and benchmark suites for benchmarking ontology building tools** provided a description of the benchmark suites that have been used in the benchmarking and how the execution of the experiments was automated for the WebODE ontology development tool.