

---

## **D1.2.2.1.2 Benchmarking the interoperability of ontology development tools using OWL as interchange language**

---

**Raúl García-Castro (UPM)**

**with contributions from:**

**Stefano David (UPM)**

**Jesús Prieto-González (UPM)**

**Abstract.**

EU-IST Network of Excellence (NoE) FP6-507482 KWEB

Deliverable D1.2.2.1.2 (WP 1.2 & WP2.1)

This deliverable describes the benchmarking of the interoperability of ontology development tools using OWL as interchange language that has taken place in Knowledge Web, including the analysis of the results obtained.

Keyword list: benchmarking, benchmark suite, interoperability, OWL

Document Identifier	KWEB/2007/D1.2.2.1.2/v1.3
Project	KWEB FP6-507482
Version	v1.3
Date	25. October, 2007
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

### **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

### **Free University of Bozen-Bolzano (FUB)**

Piazza Domenicani 3  
39100 Bolzano  
Italy  
Contact person: Enrico Franconi  
E-mail address: franconi@inf.unibz.it

### **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

### **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

### **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

### **Freie Universität Berlin (FU Berlin)**

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

### **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

### **Learning Lab Lower Saxony (L3S)**

Expo Plaza 1  
30539 Hannover  
Germany  
Contact person: Wolfgang Nejdl  
E-mail address: nejdl@learninglab.de

### **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 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 Sheffield (USFD)**

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

**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

**University of Aberdeen (UNIABDN)**

Kings College  
AB24 3FX Aberdeen  
United Kingdom  
Contact person: Jeff Pan  
E-mail address: jpan@csd.abdn.ac.uk

**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 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 Trento (UniTn)**

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

**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

Free University of Bozen-Bolzano

Universidad Politécnica de Madrid

University of Karlsruhe

University of Sheffield

Vrije Universiteit Amsterdam

---

# Changes

Version	Date	Author	Changes
0.1	08.08.07	Raúl García-Castro	First draft
0.2	13.08.07	Raúl García-Castro	Inserted the organization of the benchmarking
0.3	05.09.07	Raúl García-Castro	Inserted the interoperability results for SemTalk and WebODE
0.4	14.09.07	Raúl García-Castro	Improved the visualization of the interoperability results
0.5	16.09.07	Raúl García-Castro	Inserted the description of the IBSE tool
1.0	18.09.07	Raúl García-Castro	First version of the document sent to the Quality Assessor (Asunción Gómez-Pérez)
1.1	10.10.07	Raúl García-Castro	Included the comments from Rosario Plaza
1.2	11.10.07	Raúl García-Castro	Removed the results of the NeOn Toolkit
1.3	25.10.07	Raúl García-Castro	Included the comments from the Quality Controller (Sean Bechhofer)

# Executive Summary

In 2006, an activity for benchmarking the interoperability of ontology development tools using OWL 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 the 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 ontology dataset used in the benchmarking.
- Description of the evaluation infrastructure that automates the execution of the experiments.
- Detailed analysis of the results obtained in the benchmarking.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Benchmarking OWL interoperability</b>	<b>3</b>
2.1	Plan phase . . . . .	4
2.2	Experiment phase . . . . .	9
<b>3</b>	<b>Ontology dataset</b>	<b>14</b>
3.1	Benchmarks that depend on the knowledge model . . . . .	15
3.2	Benchmarks that depend on the syntax . . . . .	25
3.3	Description of the benchmarks . . . . .	27
3.4	Towards benchmark suites for OWL DL and OWL Full . . . . .	28
<b>4</b>	<b>The IBSE tool</b>	<b>32</b>
4.1	IBSE requirements . . . . .	32
4.2	IBSE implementation . . . . .	33
4.3	Using IBSE . . . . .	38
<b>5</b>	<b>OWL interoperability results and analysis</b>	<b>40</b>
5.1	Analysis of the import and export operation . . . . .	40
5.2	Analysis of the interoperability . . . . .	60
<b>6</b>	<b>Conclusion</b>	<b>75</b>
<b>A</b>	<b>List of benchmarks of the OWL Lite Import Benchmark Suite</b>	<b>77</b>
<b>B</b>	<b>Description of the ontologies in DL</b>	<b>90</b>
<b>C</b>	<b>The <i>benchmarkOntology</i> and <i>resultOntology</i> ontologies</b>	<b>100</b>

# Chapter 1

## Introduction

*by* RAÚL GARCÍA-CASTRO

Ontologies enable interoperability among heterogeneous applications. Ideally, ontologies defined using the W3C recommended languages (RDF(S) and OWL) should be correctly interchanged between the different tools that can manage these languages (i.e., one person should be able to develop one OWL ontology in his favourite ontology development tool and then to use this ontology in a certain annotation tool to annotate his personal web page).

Nevertheless, the current Semantic Web tools have problems in interchanging ontologies, either when these ontologies come from other tools or when they are downloaded from the web. Sometimes the problems arise because of the different representation formalisms used by the tools as not every tool natively supports RDF(S) and OWL; other times, however, the problems are caused by defects in the tools.

Not to be aware of these problems causes that the interoperability between the different Semantic Web technologies be unknown, and this is so mainly because the interoperability of the tools is not evaluated since there is no easy way of performing these evaluations.

As a previous activity in Knowledge Web, the benchmarking of the interoperability of ontology development tools was carried out using RDF(S) as interchange language [García-Castro *et al.*, 2006]. As a result, we obtained a clear picture of the RDF(S) interoperability of the tools participating in the benchmarking, namely, Corese, Jena, KAON, Sesame, Protégé, and WebODE.

In the RDF(S) Interoperability Benchmarking the experimentation and analysis of the results were performed manually. This had the advantage of obtaining high detailed results, being easier to diagnose problems in the tools and so to improve them. However, the manual execution and analysis of the results also makes the experimentation costly. Tools developers have often automated the execution of the benchmark suites but not always. Furthermore, the results obtained may be influenced by human mistakes since they depend on the people performing the experiments and on their expertise with the



tools.

As a second step, in Knowledge Web we have organised the benchmarking of the interoperability of Semantic Web technology using OWL as interchange language. This time, the goals are similar to those of the previous benchmarking activity:

- To provide mechanisms for large-scale evaluation of the interoperability of Semantic Web technology using OWL as interchange language.
- To assess and improve the current interoperability of the Semantic Web technology. This will help to know the current state of the interoperability between the tools and to correct their defects.

Although we have similar goals to those of the RDF(S) interoperability benchmarking, our approach to the benchmarking is different. The main changes performed are intended to broaden the scope of the benchmarking since we consider benchmarking any type of Semantic Web technology instead of just ontology development tools, and to automate the experiment execution and the analysis of the results.

By the time of writing this deliverable, nine tools are participating in the benchmarking: one ontology-based annotation tool: GATE; three ontology repositories: Jena, KAON2, and SWI-Prolog; and five ontology development tools: the NeOn toolkit, Protégé-Frames, Protégé-OWL, Semtalk, and WebODE.

This deliverable originated from the joint work of WP 1.2 in the industry area and of WP 2.1 in the research area. 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.

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 the deliverable.

The document is structured as follows: Chapter 2 presents how the OWL Interoperability Benchmarking was conducted following the Knowledge Web benchmarking methodology. Chapters 3 and 4 describe the ontology dataset used for the experimentation and IBSE, the evaluation infrastructure that automates the execution of the experiments, respectively. Chapter 5 includes the analysis of the interoperability, using OWL as interchange language, of the Semantic Web tools that participated in the benchmarking. And, finally, Chapter 6 draws some conclusions from the work presented in the deliverable.

## Chapter 2

# Benchmarking OWL interoperability

by RAÚL GARCÍA-CASTRO

This chapter presents how the OWL interoperability benchmarking was organized and carried out following the methodology for benchmarking ontology tools developed by the authors in the scope of Knowledge Web [García-Castro *et al.*, 2004].

The benchmarking methodology provides the general guidelines that have to be adapted to this case. 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.

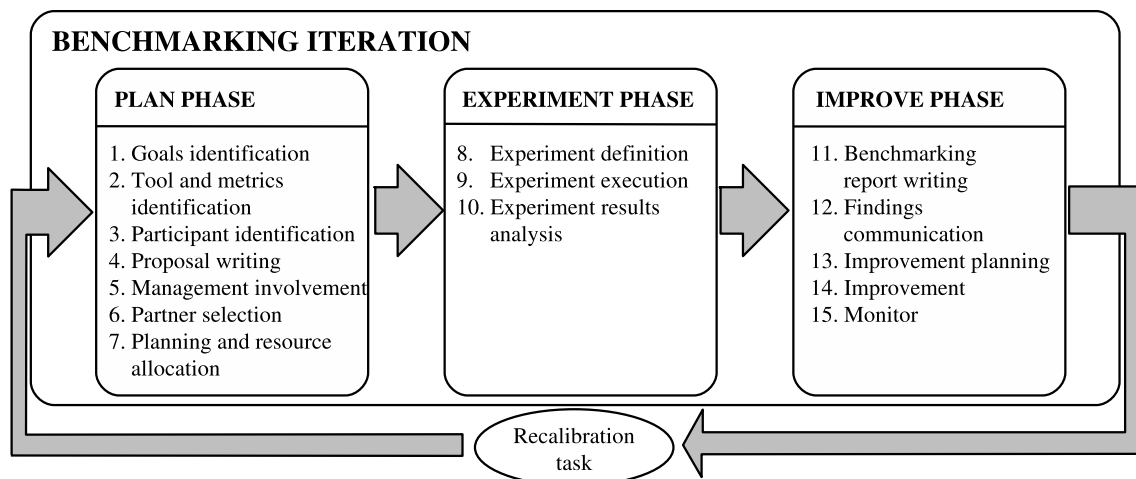


Figure 2.1: The benchmarking methodology for ontology tools

## 2.1 Plan phase

Raúl García Castro, from the UPM, assumed the role of the benchmarking initiator and organised the benchmarking; he carried out the first tasks of its process.

### 2.1.1 Benchmarking goals

According to the software benchmarking methodology, the first task to perform is to identify the benchmarking goals, benefits and costs.

The general goal of all the benchmarking activities that take place in Knowledge Web is to support the industrial applicability of Semantic Web technology. Therefore, in the benchmarking we consider any type of Semantic Web technology. In the case of the RDF(S) Interoperability Benchmarking, the scope was limited to one type of technology, namely ontology development tools.

We have focused on one problem that currently affects these tools, that of their interoperability. Achieving interoperability between Semantic Web technologies is not straightforward when these tools do not share a common knowledge model and their users do not to know the effects of interchanging an ontology from one tool to another.

Therefore, **our goal is to evaluate and improve the interoperability of Semantic Web technology.**

Other evaluation criteria could be considered when evaluating Semantic Web technology, i.e., performance, scalability, robustness, etc. In our case, we have contemplated only interoperability. An approach for benchmarking the performance and scalability of ontology development tools can be found in [García-Castro and Gómez-Pérez, 2005].

The **benefits** pursued through this goal are related to the expected outcomes of the benchmarking and involve different communities that are related to the Semantic Web tools, namely, the research community, the industrial community, and the tool developers. These benefits are:

- To create consensual processes and mechanisms for evaluating 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 these tools that affect their interoperability.
- To extract from these practices those that can be considered best practices when developing these tools.

Most of the benchmarking **expenditure** goes to the human resources needed to organise the benchmarking activity and to perform the experimentation on the tools. Other

minor expenditure goes to travelling and computers, but it is negligible when compared to the aforementioned.

### 2.1.2 Tool and metrics identification

Once we have identified the goals, benefits and costs of the benchmarking, we have to define its scope, by selecting which software from the organisation will participate in the benchmarking, which of its functionalities will be measured, and which will be the evaluation criteria to be used to assess these functionalities.

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.

As the goal presented in the previous section is too general, we have refined the scope of the benchmarking to cover a concrete interoperability scenario.

The most common way used by Semantic Web technology to interoperate and, therefore, the one that we have considered, is the indirect interchange of ontologies by storing them in a shared resource. A direct interchange of ontologies would require developing interchange mechanisms for each pair of tools, which would be very costly.

In our case, the representation formalism used for interchanging ontologies is OWL [McGuinness and van Harmelen, 2004] and the shared resource is a local filesystem where ontologies are stored in text files serialized using the RDF/XML syntax, since this is the syntax most used by Semantic Web technology.

Also, we have considered that the Semantic Web tools have different knowledge representation formalisms. In practice, it may occur that two Semantic Web tools use the same formalism or that a Semantic Web tool uses the OWL formalism.

In this scenario, interoperability depends on two different tool functionalities, the one that reads an ontology stored in the tool and writes it into an OWL file (OWL exporter from now on) and the one that reads an OWL file with an ontology and stores this ontology into the tool (OWL importer from now on).

If the evaluation criteria must describe in depth the interoperability between the tools, 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 using OWL as interchange language, we need to know:

- The components of the knowledge model of a tool that can be interchanged with another.
- The secondary effects of interchanging these components, such as insertion or loss of information.

- The subset of the knowledge models of the tools that the tools can use to correctly interoperate.
- The problems that occur when interchanging ontologies between two tools and the causes of these problems.

The delimited benchmarking scope guides when to identify the organisation members that are related to the benchmarking and when to form the benchmarking team that will be the responsible for continuing with the benchmarking in the organisation.

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.

### 2.1.3 Proposal writing

The next tasks to perform are to compile all the benchmarking-related information into a benchmarking proposal, which will be a reference along the benchmarking, and to present this proposal to the organisation management so as to obtain their approval and support.

To reach a broader audience, the benchmarking proposal did not take the form of a paper document but of a publicly available web page<sup>1</sup>.

This web page includes all the relevant information about the benchmarking and is updated as the benchmarking advances. Currently, the information included in the web page is the following:

- Motivation.
- Goals.
- Benefits and costs.
- Tools and people involved.
- Description of the experimentation.
- Benchmark suite.
- Planning.
- Related events.
- Results and recommendations.

---

<sup>1</sup>[http://knowledgeweb.semanticweb.org/benchmarking\\_interoperability/owl/](http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/)

This benchmarking proposal was presented to the manager of the Ontology Engineering Group and, after her analysis, she agreed on the continuity of the benchmarking and on the allocation of future resources both for performing the experimentation and for improving the tool.

## 2.1.4 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 OWL 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 OWL, which were found by the time of performing this task (April 2007). Their developers were directly contacted.

Tool	Institution	URL
Altova Semanticworks	Altova	<a href="http://www.altova.com/products/semanticworks/">http://www.altova.com/products/semanticworks/</a>
DOE	Inst. National de l'Audiovisuel	<a href="http://homepages.cwi.nl/troncy/DOE/">http://homepages.cwi.nl/troncy/DOE/</a>
DOME	DERI	<a href="http://dome.sourceforge.net/">http://dome.sourceforge.net/</a>
GrOWL	University of Vermont	<a href="http://ecoinformatics.uvm.edu/technologies/index.html">http://ecoinformatics.uvm.edu/technologies/index.html</a>
Hozo	Osaka University	<a href="http://www.ei.sanken.osaka-u.ac.jp/hozo/eng/index_en.php">http://www.ei.sanken.osaka-u.ac.jp/hozo/eng/index_en.php</a>
IBM IODT	IBM	<a href="http://www.alphaworks.ibm.com/tech/semanticstk">http://www.alphaworks.ibm.com/tech/semanticstk</a>
KAON2	Universitat Karlsruhe	<a href="http://kaon2.semanticweb.org/">http://kaon2.semanticweb.org/</a>
Linkfactory Workbench	Language & Computing	<a href="http://www.landcglobal.com/pages/linkfactory.php">http://www.landcglobal.com/pages/linkfactory.php</a>
m3t4 Studio	Metatomix	<a href="http://www.m3t4.com/">http://www.m3t4.com/</a>
Medius Visual O. M.	Sandpiper Software	<a href="http://www.sandsoft.com/products.html">http://www.sandsoft.com/products.html</a>
Model Futures OWL Editor	Model Futures	<a href="http://www.modelfutures.com/OwlEditor.html">http://www.modelfutures.com/OwlEditor.html</a>
The NeOn Toolkit	The NeOn project	<a href="http://www.neon-toolkit.org/">http://www.neon-toolkit.org/</a>
OntoTrack	University of Ulm	<a href="http://www.informatik.uni-ulm.de/ki/ontotrack/">http://www.informatik.uni-ulm.de/ki/ontotrack/</a>
Powl	University of Leipzig	<a href="http://aksw.informatik.uni-leipzig.de/Projects/Powl">http://aksw.informatik.uni-leipzig.de/Projects/Powl</a>
Protégé-Frames	Stanford University	<a href="http://Protégé.stanford.edu/">http://Protégé.stanford.edu/</a>
Protégé-OWL	University of Manchester	<a href="http://Protégé.stanford.edu/">http://Protégé.stanford.edu/</a>
SemTalk	Semtation	<a href="http://www.semtalk.com/">http://www.semtalk.com/</a>
SWOOP	University of Maryland	<a href="http://www.mindswap.org/2004/SWOOP/">http://www.mindswap.org/2004/SWOOP/</a>
Topbraid Composer	TopQuadrant	<a href="http://www.topbraidcomposer.com/">http://www.topbraidcomposer.com/</a>
VisioOWL	John Flynn	<a href="http://mysite.verizon.net/jflynn12/VisioOWL/VisioOWL.htm">http://mysite.verizon.net/jflynn12/VisioOWL/VisioOWL.htm</a>
WebODE	U. Politécnica de Madrid	<a href="http://webode.dia.fi.upm.es/WebODEWeb/index.html">http://webode.dia.fi.upm.es/WebODEWeb/index.html</a>

Table 2.1: Ontology development tools capable of importing/exporting OWL

Tool	Version	Developer	Experimenter
GATE	4.0	Sheffield U.	Sheffield U.
Jena	2.3	HP	U. Politécnica de Madrid
KAON2	2006-09-22	Karlsruhe U.	Karlsruhe U.
NeOn Toolkit	1.0 build 823	The NeOn project	The NeOn project
Protégé	3.3 build 395	Stanford U.	CERTH
Protégé-OWL	3.3 build 395	Manchester U.	CERTH
SemTalk	2.3	Semtation	Semtation
SWI-Prolog	5.6.35	U. of Amsterdam	U. of Amsterdam
WebODE	2.0 build 140	U. Politécnica de Madrid	U. Politécnica de Madrid

Table 2.2: Semantic Web tools participating in the benchmarking

Any Semantic Web tool capable of importing and exporting OWL can participate in the benchmarking. Table 2.2 shows the nine tools that are taking part in the benchmarking when writing this deliverable: one ontology-based annotation tool: GATE<sup>2</sup>; three ontology repositories: Jena<sup>3</sup>, KAON2<sup>4</sup>, and SWI-Prolog<sup>5</sup>; and five ontology development tools: the NeOn toolkit<sup>6</sup>, Protégé-Frames<sup>7</sup>, Protégé-OWL<sup>8</sup>, Semtalk<sup>9</sup>, and WebODE<sup>10</sup>.

The experimentation over the NeOn Toolkit has been performed in the scope of the NeOn European project<sup>11</sup> and the analysis of the NeOn Toolkit interoperability is presented in [García-Castro, 2007]. The results of this interoperability are not included in this deliverable as they are restricted to the NeOn partners.

The conclusions reached about some of these tools could be applied to other tools that use the same mechanisms for managing ontologies as the ones used by these tools. For instance, the KIM<sup>12</sup> ontology-based annotation tool has the same representation formalism and uses the same ontology management API as GATE. Hence, it is expected that the interoperability results of KIM are identical to those of GATE and, therefore, experiments have not been performed over KIM.

---

<sup>2</sup><http://gate.ac.uk/>

<sup>3</sup><http://jena.sourceforge.net/>

<sup>4</sup><http://kaon2.semanticweb.org/>

<sup>5</sup><http://www.swi-prolog.org/packages/semweb.html>

<sup>6</sup><http://www.neon-toolkit.org/>

<sup>7</sup><http://protege.stanford.edu/>

<sup>8</sup><http://protege.stanford.edu/overview/protege-owl.html>

<sup>9</sup><http://www.semtalk.com/>

<sup>10</sup><http://webode.dia.fi.upm.es/>

<sup>11</sup><http://www.neon-project.org/>

<sup>12</sup><http://www.ontotext.com/kim/>

## 2.1.5 Planning and resource allocation

The main deadline of the benchmarking was imposed by the deadline of this Knowledge Web deliverable. Therefore, we designed a plan that included the *Plan* and *Experiment* phases, though it just included the first task of the *Improve* phase (Benchmarking report writing).

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.

## 2.2 Experiment phase

### 2.2.1 Experiment definition

The design principles taken into account when developing the experimentation and the benchmark suite are related to the main desirable properties that a benchmark suite must have and that have been stated by many different authors [Sim *et al.*, 2003, Bull *et al.*, 1999, Shirazi *et al.*, 1999, Stefani *et al.*, 2003]: accessibility, affordability, simplicity, representativity, portability, scalability, robustness, and consensus.

The experiments to be performed in the benchmarking must provide data informing how the Semantic Web tools comply with the evaluation criteria defined in the previous section:

- The components of the knowledge model of a tool that can be interchanged with another.
- The secondary effects of interchanging these components, such as insertion or loss of information.
- The subset of the knowledge models of the tools that these tools can use to correctly interoperate.
- The problems that occur when interchanging ontologies between two tools and the causes of these problems.

Interoperability using an interchange language depends on the capabilities of the tools to import ontologies from the language (to read one file with an ontology and to store this ontology in the tool knowledge model) and to export ontologies to the language (to write into a file an ontology stored in the tool knowledge model). Therefore, the experimentation provided data not only about the interoperability but also about the OWL importers and exporters of the tools.



As we mentioned before, participation in the benchmarking is open to any Semantic Web tool. Nevertheless, the experimentation requires that the tools participating be able to import and export OWL ontologies. This is because in the experimentation, we need an automatic and uniform way of accessing the tools and the operations performed to access the tools must be supported by most of the Semantic Web tools. Because of the high heterogeneity in Semantic Web tools, ontology management APIs vary from one tool to another. Therefore, the way chosen to automatically access the tools is through the following two operations commonly supported by most Semantic Web tools: to import one ontology from a file, and to export one ontology into a file.

During the experiment, a common group of benchmarks is executed and each benchmark describes one input ontology that has to be interchanged between a single tool and the others (including itself).

Each benchmark execution comprises two sequential steps, shown in Figure 2.2. Starting with a file that contains an ontology ( $O1$ ), the first step (*Step 1*) consists in importing the file with the ontology into the origin tool and then exporting the ontology into a file using the interchange language ( $O1''$ ). The second step (*Step 2*) consists in importing the file with the ontology exported by the origin tool ( $O1''$ ) into the destination tool and then exporting the ontology into another file ( $O1'''$ ).

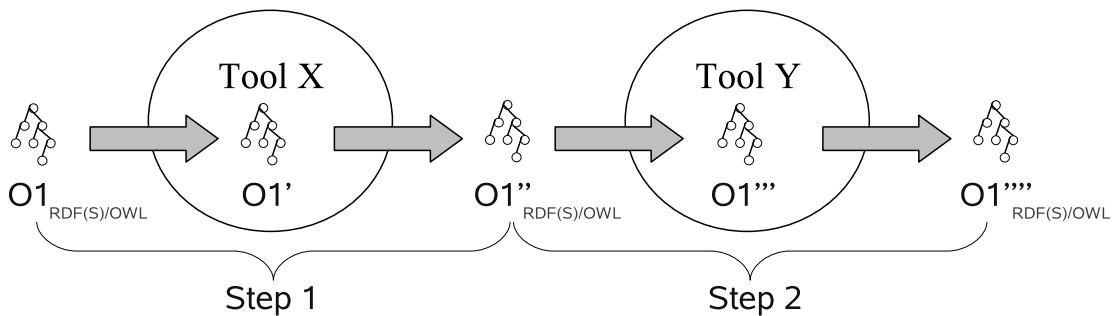


Figure 2.2: The two steps of a benchmark execution

In these steps, there is not a common way for the tools to check the results of importing the ontologies ( $O1'$  and  $O1'''$ ), we just have the results of combining the import and export operations (the files exported by the tools), so we consider these two operations as an atomic operation. It must be noted, therefore, that if a problem arises in one of these steps, we cannot know whether the problem was originated when importing or when exporting the ontology since we do not know the state of the ontology inside each tool.

After a benchmark execution, the results obtained from the ontology described in the benchmark are three different states, namely, the original ontology ( $O1$ ), the intermediate ontology exported by the first tool ( $O1''$ ), and the final ontology exported by the second tool ( $O1'''$ ). From these results, we define the evaluation criteria for a benchmark execution. These evaluation criteria will be considered in *Step 1*, *Step 2*, and in the whole interchange (*Step 1 + Step 2*); they are the following:

- **Execution** (*OK/FAIL/C.E./N.E.*) informs of the correct execution of a step or the whole interchange. Its value is *OK* if the step or the whole interchange is carried out with no execution problem; *FAIL* if the step or the whole interchange is carried out with some execution problem; *C.E.* (Comparer Error) if the comparer launches an exception when comparing the original and the final ontologies; and *N.E.* (Not Executed) if the second step is not executed because the execution on the first step failed.
- **Information added or lost** informs of the information added to or lost from the ontology in terms of triples in each step or in the whole interchange. We can know the triples added or lost in *Step 1*, in *Step 2*, and in the whole interchange by comparing the original ontology with the intermediate one, then the intermediate ontology with the final one, and the original with the final ontology, respectively.
- **Interchange** (*SAME/DIFFERENT/NO*) informs whether the ontology has been interchanged correctly with no addition or loss of information. From the previous basic measurements, we can define *Interchange* as a derived measurement that is *SAME* if *Execution* is *OK* and *Information added* and *Information lost* are void; *DIFFERENT* if *Execution* is *OK* but *Information added* or *Information lost* are not void; and *NO* if *Execution* is *FAIL*, *N.E.* or *C.E.*.

The experiment described above could use as input ontologies described in any formalism (RDF(S), OWL, etc.). Nevertheless, following the goals of the benchmarking, we use OWL ontologies as input and as interchange. Also, these ontologies must be serialized in the RDF/XML syntax, as this is the most commonly used by the tools for interchanging ontologies.

Another issue is which ontologies to use for evaluating the interoperability of the tools. Any group of ontologies could be used in the experimentation, but using real, large or complex ontologies can be useless if we do not know whether the tools can interchange simple ontologies correctly. Because one of the goals of the benchmarking is to improve the tools, the ontologies must be simple to isolate problem causes and to identify possible problems.

Therefore, the OWL Lite Import Benchmark Suite<sup>13</sup> was used for evaluating the interoperability of the tools; this benchmark suite is common for all the tools and contains ontologies with simple combinations of the OWL knowledge model. The complete description and the procedure followed to define this benchmark suite can be found in Section 3.

The quality of the benchmark suite to be used is essential for the results of the benchmarking. Therefore, once the benchmark suite was defined, it was published on the benchmarking web page so that they could be reviewed by the participants. It was also presented and discussed in several Knowledge Web meetings.

<sup>13</sup>[http://knowledgeweb.semanticweb.org/benchmarking\\_interoperability/owl/import.html](http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/import.html)

The experiments to perform in the benchmarking consist in interchanging each of the ontologies of the OWL Lite Import Benchmark Suite between all the tools (including interchanges from one tool to itself) and in collecting the results of these interchanges.

Although the results of the experiment described above could be obtained manually, the goal of the benchmarking is to automate all the experimentation. Hence, we need some software application that performs all the experiments automatically.

This software application is IBSE<sup>14</sup> (Interoperability Benchmark Suite Executor) and will be in charge of executing the experiments and of generating visualizations of the results of these experiments. A description of the IBSE tool and the specific procedure to follow for using it are detailed in Chapter 4.

### 2.2.2 Experimentation planning

The planning of the benchmarking was defined so as the deadlines would coincide with the Knowledge Web deadline when the benchmarking results should be delivered. Therefore, a plan was designed that included the *Plan* and *Experiment* phases, though it just included the first task of the *Improve* phase (*Benchmarking report writing*).

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.

The planning for the experimentation included the following steps:

1. To develop the IBSE tool.
2. To adapt the IBSE tool to the tools participating in the benchmarking.
3. To execute the experiments.
4. To analyse the results.

### 2.2.3 Experiment execution and result analysis

Once the IBSE tool was adapted to include all the tools participating in the benchmarking, the experiments were automatically performed. As mentioned in Section 2.1.4, we obtained interoperability results for nine tools: GATE, Jena, KAON2, the NeOn Toolkit, Protégé-Frames, Protégé-OWL, SemTalk, SWI-Prolog, and WebODE.

---

<sup>14</sup><http://knowledgeweb.semanticweb.org/benchmarking-interoperability/ibse/>

Raúl García-Castro compiled all the execution results, made them available in the benchmarking web page<sup>15</sup>, and provided a general interpretation of them, shown in Chapter 5.

---

<sup>15</sup>[http://knowledgeweb.semanticweb.org/benchmarking\\_interoperability/owl/2007-08-12\\_Results/htmls/](http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/2007-08-12_Results/htmls/)

# Chapter 3

## Ontology dataset

by STEFANO DAVID AND RAÚL GARCÍA CASTRO

As we mentioned in the previous chapter, any group of ontologies could be used as input for the experiment. For example, we could employ a group of real ontologies in a certain domain, such as ontologies synthetically generated such as the Lehigh University Benchmark (LUBM) [Guo *et al.*, 2005], the University Ontology Benchmark (UOB) [Ma *et al.*, 2006], and the OWL Test Cases<sup>1</sup> (developed by the W3C Web Ontology Working Group).

These ontologies were designed with specific goals and requirements. Real ontologies are developed to represent knowledge in some application; the LUBM and the UOB aim to evaluate the performance of the tools under certain circumstances, and the OWL Test Cases check if a tool deals correctly with the OWL language, clarify the formal meaning of the constructors and show examples of their use.

However, as our goal was to improve interoperability, these ontologies could complement our experiments even though they were designed with specific goals and requirements such as these of performance or correctness evaluation. In our case, we aim to evaluate interoperability with simple OWL ontologies that, although they do not cover exhaustively the OWL specification, allow highlighting problems in the tools.

To this end, we have defined the OWL Lite Import Benchmark Suite [David *et al.*, 2006]. This benchmark suite was intended to evaluate the OWL import capabilities of Semantic Web tools, but we now use it to evaluate the interoperability of Semantic Web tools by checking the interchange of ontologies with simple combinations of components of the OWL Lite knowledge model.

The assumptions concerning the development of the OWL Lite Import Benchmark Suite, are the following:

- The number of benchmarks should be small. Benchmarking is a process that consumes a lot of resources, and any increase in the number of benchmarks leads to an

---

<sup>1</sup><http://www.w3.org/TR/owl-test/>

increment in the time required for performing the experiments and for the subsequent analysis of the results.

- The use of OWL Lite to define the ontologies and so to limit the number of benchmarks. Furthermore, we do not consider annotation, versioning and heading vocabulary terms.
- To use the RDF/XML syntax<sup>2</sup> for writing OWL ontologies since this syntax is the most used by Semantic Web tools for importing and exporting ontologies.
- To define correct ontologies only. The ontologies defined in the benchmarks do not contain syntactic or semantic errors and, in order to ensure the syntactic correctness of the ontologies, we decided to use an OWL validator<sup>3</sup>.
- To define simple ontologies only. This will allow to easily detecting problems in the tools.

There are two different issues that affect the correct import of an ontology: a) which combinations of components of the OWL knowledge model are present in the ontology; and b) which of the different variants of the RDF/XML syntax are present in the ontology. Therefore, to isolate each of these issues, we have defined separately the benchmarks that depend on the OWL knowledge model and those that depend on the OWL syntax chosen. To increase the usability of the benchmarks, they also have been divided in groups.

The next sections explain how these two types of benchmarks have been defined.

### 3.1 Benchmarks that depend on the knowledge model

The process we followed to define the ontologies contained in the benchmarks was the following: we first defined the ontologies in natural language, then we expressed them in the OWL abstract syntax using the productions, and finally we wrote them in the RDF/XML syntax.

In the definition of the ontologies, we considered the different possibilities of defining in OWL classes (with a class identifier, with a value or cardinality restriction on a property, or with the intersection operator), properties (object and datatype properties with range, domain, and cardinality constraints, relations between properties, global cardinality constraints, and logical property characteristics), and instances (with named and anonymous individuals, equivalence and differences among individuals).

Moreover, we decided to discard those vocabulary terms that do not contribute to the OWL expressiveness; these are the annotation, versioning, and heading vocabulary terms.

---

<sup>2</sup><http://www.w3.org/TR/rdf-syntax-grammar/>

<sup>3</sup><http://phoebus.cs.man.ac.uk:9999/OWL/Validator>

We considered at most one or two OWL vocabulary terms at a time, and then we studied all the possible combinations of these terms with the remaining. When the number of the ontologies defined was large, we pruned the benchmark suite. We also decided to consider the combinations of the OWL vocabulary terms with a cardinality of zero, one, and two, assuming that the result for higher cardinalities equals the result for cardinality two.

The reminder of this section presents the ontologies defined for the benchmarks in each group, along with the vocabulary terms and the productions (axioms) involved.

The conventions used in the productions are those used in the OWL specification<sup>4</sup>, i.e., a start symbol of the language is capitalized, otherwise it is lowercase; terminals are quoted; alternatives are separated by a colon (|) or given in different productions; square brackets ([. . .]) indicate elements that occur at most once; and braces ({. . .}) indicate elements that can occur any number of times, including zero.

### 3.1.1 Benchmarks for classes

In OWL Lite, classes can be described by a class identifier, by a value or a cardinality restriction on a property, or by the intersection operator. From these building blocks, we used the OWL Lite class and restriction axioms and defined the different ways of describing a class in OWL Lite with these axioms.

We decided to group the benchmarks according to the following criteria: classes and class hierarchies, class equivalences, and classes defined using a set operator.

#### Group A: Classes and class hierarchies

The ontologies of this group describe classes and class hierarchies. This group includes classes that are a subclass of value restrictions, cardinality restrictions on properties, and class intersections.

In this group, we focus on vocabulary terms of both RDF(S) and OWL<sup>5</sup>:

```
rdfs:subClassOf, owl:Class, owl:Restriction, owl:onProperty,  
owl:someValuesFrom, owl:allValuesFrom, owl:cardinality,  
owl:maxCardinality, owl:minCardinality, owl:intersectionOf
```

The productions we used for defining the benchmarks are:

```
axiom ::= 'Class('classID modality  
         {super}')'
```

---

<sup>4</sup><http://www.w3.org/TR/owl-semantic/syntax.html>

<sup>5</sup>In boldface we highlight the main vocabulary terms of the group.

```

modality ::= 'partial'
super ::= classID | restriction
restriction ::= 'restriction(' datavaluedPropertyID
                dataRestrictionComponent ')'
                | 'restriction(' individualvaluedPropertyID
                individualRestrictionComponent ')'
dataRestrictionComponent ::= 'allValuesFrom(' dataRange ')'
                | 'someValuesFrom(' dataRange ')'
                | cardinality
individualRestrictionComponent ::= 'allValuesFrom(' classID ')'
                | 'someValuesFrom(' classID ')'
                | cardinality
cardinality ::= 'minCardinality(0)' | 'minCardinality(1)'
                | 'maxCardinality(0)' | 'maxCardinality(1)'
                | 'cardinality(0)' | 'cardinality(1)'
dataRange ::= datatypeID | 'rdfs:Literal'
datatypeID ::= URIreference
classID ::= URIreference
datavaluedPropertyID ::= URIreference
individualvaluedPropertyID ::= URIreference

```

To see how the productions of the OWL abstract syntax are used in the definition of the OWL ontologies, let's consider the ontology of benchmark *ISA07*. This ontology contains a class (e.g., *Driver*), which is subclass of an anonymous class defined by an *owl:someValuesFrom* value restriction in the object property *hasCar*, which can have only instances of class *Car* as range.

In the abstract syntax, we can express this ontology as follows:

```

Ontology( <http://www.example.org/ISA07.owl>
  ObjectProperty(myNs:hasCar)
  Class(myNs:Car partial)
  Class(myNs:Driver partial
    restriction(myNs:hasCar someValuesFrom(myNs:Car)))
)

```

The ontology is written in the RDF/XML syntax as follows:

```

<owl:Ontology rdf:about="#" />
<owl:ObjectProperty rdf:about="&myNs;hasCar"/>
<owl:Class rdf:about="&myNs;Driver">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&myNs;hasCar"/>
      <owl:someValuesFrom>
        <owl:Class rdf:about="&myNs;Car" />

```



```
        </owl:someValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

### Group B: Class equivalence

The ontologies of this group describe class equivalences. These are classes equivalent to value and cardinality restrictions on properties and classes equivalent to intersection of classes. Moreover, both this group and group A are intended to test the ability of the tools in coping with the difference between a subclass relation and an equivalent class relation. The benchmarks of this group are alike those in Group A; the only difference is that Group A contains primitive classes (with *modality* = '*partial*') and Group B contains defined classes (with *modality* = '*complete*').

In this group, the vocabulary terms concerned are:

**owl:equivalentClass**, **owl:Class**, owl:Restriction, owl:onProperty,  
owl:someValuesFrom, owl:allValuesFrom, owl:cardinality,  
owl:maxCardinality, owl:minCardinality, owl:intersectionOf

The productions we used for defining the benchmarks are:

```
axiom ::= 'Class('classID modality
          {super}')'
axiom ::= 'EquivalentClasses('classID classID {classID}''
modality ::= 'complete'
super ::= classID | restriction | description
restriction ::= 'restriction('datavaluedPropertyID
                dataRestrictionComponent')'
                | 'restriction('individualvaluedPropertyID
                individualRestrictionComponent')'
dataRestrictionComponent ::= 'allValuesFrom('dataRange')'
                          | 'someValuesFrom('dataRange')'
                          | cardinality
individualRestrictionComponent ::= 'allValuesFrom('classID')'
                                | 'someValuesFrom('classID')'
                                | cardinality
cardinality ::= 'minCardinality(0)' | 'minCardinality(1)'
               | 'maxCardinality(0)' | 'maxCardinality(1)'
               | 'cardinality(0)'    | 'cardinality(1)'
dataRange ::= datatypeID | 'rdfs:Literal'
datatypeID ::= URIreference
classID ::= URIreference
datavaluedPropertyID ::= URIreference
individualvaluedPropertyID ::= URIreference
```

### Group C: Class defined with set operators

The ontologies defined in this group describe classes that are defined by set operators. Although the OWL language has three vocabulary terms for expressing set operations (i.e., *owl:unionOf*, *owl:intersectionOf*, and *owl:complementOf*, which correspond to logical disjunction, conjunction, and negation respectively), OWL Lite can only express classes that are intersection of other classes.

In this group, the vocabulary terms concerned are:

**owl:intersectionOf**, **owl:Class**

The production we used for defining these benchmarks are:

```
axiom ::= 'Class('classID modality {super}{'')'
modality ::= 'complete'|'partial'
super ::= classID
classID ::= URIreference
```

### 3.1.2 Benchmarks for properties

In OWL Lite, properties can be either object properties (properties that link a class with another class) or datatype properties (properties that link a class with a data value).

We grouped the benchmarks of this group according to the following criteria: description of properties and property hierarchies, properties with domain and range, relations between properties, and global cardinality constraints and logical characteristics of properties.

#### Group D: Property and property hierarchies

The ontologies of this group describe properties and property hierarchies.

In this group, the vocabulary terms concerned are:

**owl:ObjectProperty**, **owl:DatatypeProperty**, **rdfs:subPropertyOf**.

The axioms of the abstract syntax used in this group are:

```
axiom ::= 'DatatypeProperty('datavaluedPropertyID
          {'super('datavaluedPropertyID')'})'
        | 'ObjectProperty('individualvaluedPropertyID
          {'super('individualvaluedPropertyID')'})'
datatypeID ::= URIreference
datavaluedPropertyID ::= URIreference
individualvaluedPropertyID ::= URIreference
```

### Group E: Property with domain and range

The ontologies of this group describe properties that have from one to three domain and/or range constraints. In this group we do not consider properties with no range and domain constraint since they are included in Group D.

In this group, the vocabulary terms concerned are:

```
owl:Class, owl:ObjectProperty, owl:DatatypeProperty,  
rdfs:range, rdfs:domain, rdfs:Literal.
```

The axioms of the abstract syntax are:

```
axiom ::= 'DatatypeProperty('datavaluedPropertyID  
        {'domain('classID')'}{'range('dataRange')'})'  
        | 'ObjectProperty('individualvaluedPropertyID  
        {'domain('classID')'}{'range('classID')'})'  
datatypeID ::= URIreference  
classID ::= URIreference  
datavaluedPropertyID ::= URIreference  
individualvaluedPropertyID ::= URIreference
```

### Group F: Relation between properties

The ontologies of this group describe equivalences among object properties and among datatype properties; they also describe object properties that are inverse one from the other. It is not possible to define the inverse of a datatype property, since the inverse relation would have a literal (i.e., a data value) as its domain, and this is not allowed in OWL Lite.

In this group, the vocabulary terms concerned are:

```
owl:Class, owl:ObjectProperty, owl:DatatypeProperty,  
rdfs:range, rdfs:domain, rdfs:Literal,  
owl:equivalentProperty, owl:inverseOf.
```

In this group we use the following axioms:

```
axiom ::= 'DatatypeProperty('datavaluedPropertyID  
        {'domain('classID')'}{'range('dataRange')'})'  
        | 'ObjectProperty('individualvaluedPropertyID  
        {'domain('classID')'}{'range('classID')'})'  
        [ 'inverseOf('individualvaluedPropertyID')' ]  
axiom ::= 'EquivalentProperties('datavaluedPropertyID  
        datavaluedPropertyID
```

```

                                {datavaluedPropertyID}')')'
    | 'EquivalentProperties('individualvaluedPropertyID
                                individualvaluedPropertyID
                                {individualvaluedPropertyID}')')'
dataRange ::= datatypeID | 'rdfs:Literal'
datatypeID ::= URIreference
classID ::= URIreference
datavaluedPropertyID ::= URIreference
individualvaluedPropertyID ::= URIreference

```

### Group G: Global cardinality constraints and logical characteristics of properties

In OWL, object and datatype properties can be further described with more expressive characteristics. The ontologies of this group describe properties with domain and range, which are also symmetric, transitive, functional, or inverse functional. Datatype properties can be specified only as functional, since the other specifications would lead to have literals in the domain of the datatype property, which is forbidden in OWL Lite.

In this group, the vocabulary terms concerned are:

```

owl:Class, owl:ObjectProperty, owl:DatatypeProperty,
rdfs:range, rdfs:domain, rdfs:Literal,
owl:SymmetricProperty, owl:TransitiveProperty,
owl:FunctionalProperty, owl:InverseFunctionalProperty.

```

The axiom used for generating ontologies in this group are:

```

axiom ::= 'DatatypeProperty('datavaluedPropertyID {['Functional']}
          {'domain('classID')'}) {'range('dataRange')'})')'
    | 'ObjectProperty('individualvaluedPropertyID
          ['inverseOf('individualvaluedPropertyID')'])
          ['Functional' | 'InverseFunctional' |
           'Functional' 'InverseFunctional' |
           'Transitive'] ['Symmetric']
          {'domain('classID')'}) {'range('classID')'})')')'
dataRange ::= datatypeID | 'rdfs:Literal'
datatypeID ::= URIreference
classID ::= URIreference
datavaluedPropertyID ::= URIreference
individualvaluedPropertyID ::= URIreference

```

### 3.1.3 Benchmarks for instances

In OWL Lite, individuals (named or anonymous) are instances of classes related by properties to other individuals. There are also special built-in properties for asserting relation-

ships among them.

### Group H: Single individuals

The easiest way to describe an individual is to instantiate a class: the ontologies of this group define one or more classes with single or multiple individuals as instances.

In this group the only vocabulary terms concerned are *owl:Class* and *rdf:type*. The OWL Lite axioms used in this group are:

```
axiom ::= 'Class('classID')'
fact   ::= individual
individual ::= 'Individual('[individualID] {'type('type')'}
               {value}'))'
value   ::= 'value('individualvaluedPropertyID individualID '))'
          | 'value('individualvaluedPropertyID individual '))'
          | 'value('datavaluedPropertyID dataLiteral '))'
type    ::= classID
datatypeID ::= URIreference
classID  ::= URIreference
datavaluedPropertyID ::= URIreference
individualvaluedPropertyID ::= URIreference
individualID ::= URIreference
```

### Group I: Named individual and properties

Individuals can be related one each other through user defined properties. In this group, every ontology has one object or datatype property whose domain and range are classes, and individuals as instance of these classes. Moreover, the object and datatype properties are simple (there are no logical characteristics of properties specified) and, in the case of datatype properties, there are also data values (we only used strings).

The vocabulary terms are used for defining classes and properties with range and domain constraints. The individuals are instances of these classes and properties.

**owl:Class**, **rdf:type**, owl:ObjectProperty, owl:DatatypeProperty,  
rdfs:range, rdfs:domain, rdfs:Literal.

The axioms covered in this group are:

```
axiom ::= 'Class('classID')'
        | 'DatatypeProperty(' datavaluedPropertyID
          {'domain(' classID ' ')}{'range(' dataRange ' ')}')'
        | 'ObjectProperty(' individualvaluedPropertyID
          {'domain(' classID ' ')}{'range(' classID ' ')}')'
```

```

fact ::= individual
individual ::= 'Individual('[individualID] {'type('type')'}
              {value}')'
value ::= 'value('individualvaluedPropertyID individualID ')'
        | 'value('individualvaluedPropertyID individual ')'
        | 'value('datavaluedPropertyID dataLiteral ')'
type ::= classID
datatypeID ::= URIreference
classID ::= URIreference
datavaluedPropertyID ::= URIreference
individualvaluedPropertyID ::= URIreference
individualID ::= URIreference

```

### Group J: Anonymous individuals and properties

Individuals in OWL can also be anonymous, i.e., we can refer to them without giving them an explicit name, but they can be used in assertions.

The vocabulary terms are used for defining classes and properties with range and domain constraint.

```

owl:Class, rdfs:range, rdfs:domain, rdf:type, rdfs:Literal,
owl:ObjectProperty, owl:DatatypeProperty

```

In this group the OWL Lite axioms concerned are:

```

axiom ::= 'Class('classID')'
        | 'DatatypeProperty('datavaluedPropertyID
          {'domain('classID')'}{'range('dataRange')'})'
        | 'ObjectProperty('individualvaluedPropertyID
          {'domain('classID')'}{'range('classID')'})'
fact ::= individual
individual ::= 'Individual('[individualID] {'type('type')'}
              {value}')'
value ::= 'value('individualvaluedPropertyID individualID')'
        | 'value('individualvaluedPropertyID individual')'
        | 'value('datavaluedPropertyID dataLiteral')'
type ::= classID
datatypeID ::= URIreference
classID ::= URIreference
datavaluedPropertyID ::= URIreference
individualvaluedPropertyID ::= URIreference
individualID ::= URIreference

```

**Group K: Individual identity**

The OWL vocabulary also contains built-in predicates (i.e., terms) that express basic relations among individuals. These terms can be used to state that two individuals can either be the same or different and to state that in a set of individuals, each of them is different from the others.

The vocabulary terms are used for defining classes and properties with range and domain constraint.

```
owl:Class, owl:ObjectProperty, owl:DatatypeProperty, rdfs:range,  
rdfs:domain, rdfs:Literal, rdf:type, owl:differentFrom,  
owl:sameAs, owl:AllDifferent, owl:distinctMembers
```

In this group the axioms concerned are:

```
axiom ::= 'Class('classID')'  
fact ::= 'SameIndividual('individualID  
        individualID  
        {individualID}')'  
      | 'DifferentIndividuals('individualID  
        individualID  
        {individualID}')'  
fact ::= individual  
individual ::= 'Individual('[individualID] {'type('type')'}  
              {value}'))'  
value ::= 'value('individualvaluedPropertyID individualID')'  
        | 'value('individualvaluedPropertyID individual')'  
        | 'value('datavaluedPropertyID dataLiteral')'  
type ::= classID  
datatypeID ::= URIreference  
classID ::= URIreference  
datavaluedPropertyID ::= URIreference  
individualvaluedPropertyID ::= URIreference  
individualID ::= URIreference
```

The reader can note that there is not a explicit production that generates the vocabulary terms *owl:AllDifferent* and *owl:distinctMembers*. The abstract syntax of OWL allows producing only pairwise disjoint individuals, and these two vocabulary terms are, indeed, intended as a shortcut for expressing that, given a set of individuals, each of them is unique and different from all the others in the set.

## 3.2 Benchmarks that depend on the syntax

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

These syntactic variants are the same as those considered in the RDF(S) Import Benchmark Suite. However, the ontologies defined in each benchmark suite are different since in one case they are written in RDF(S) and in the other in OWL. The benchmarks that depend on the syntax form a group on their own (Group L).

These benchmarks are arranged into different categories, each of which checks one different aspect of the possible RDF/XML variants.

**URI references.** There are different possibilities, listed below, to refer to a resource on the web. For each of them we have defined a benchmark.

- Using an absolute URI reference.

```
...
<rdf:Description
    rdf:about="http://www.example.org/ontology#Man"/>
...
```

- Using an URI reference relative to a base URI.

```
...
xml:base="http://www.example.org/ontology#"
...
<rdf:Description rdf:about="#Man" />
...
```

- Using an URI reference transformed from *rdf:ID* attribute values.

```
...
<rdf:Description rdf:ID="Man"/>
...
```

- Using an URI reference relative to an *ENTITY* declaration.

```
...
<!ENTITY myNs "http://www.example.org/ontology#">
...
xmlns:myNs="http://example.org/ontology#">
...
<rdf:Description rdf:about="&myNs;Man" />
...
```

**Abbreviations.** There are cases in which the RDF/XML syntax allows grouping statements with a same subject or shortening the RDF/XML code. We consider here bench-



marks for empty nodes, multiple properties, typed nodes, string literals, and blank nodes. For each subcategory we have defined two benchmarks.

- Empty nodes. The following two descriptions of *Woman* define exactly the same concept, but the second is written more compactly.

```
<rdf:Description rdf:about="#Woman">
  <rdf:type>
    <rdf:Description rdf:about="&owl;Class">
      </rdf:Description>
    </rdf:type>
  </rdf:Description>
</rdf:Description>
```

---

```
<rdf:Description rdf:about="#Woman">
  <rdf:type rdf:resource="&owl;Class" />
</rdf:Description>
```

- Resources with multiple properties. The following example shows how to group statements related to a resource.

```
<owl:DatatypeProperty rdf:about="#hasName">
  <rdfs:domain rdf:resource="&myNs;Person" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="#hasName">
  <rdfs:range rdf:resource="&rdfs;Literal" />
</owl:DatatypeProperty>
```

---

```
<owl:DatatypeProperty rdf:about="#hasName">
  <rdfs:domain rdf:resource="&myNs;Person" />
  <rdfs:range rdf:resource="&rdfs;Literal" />
</owl:DatatypeProperty>
```

- Typed nodes. They can be expressed in two equivalent ways:

```
<rdf:Description rdf:about="#Man">
  <rdf:type rdf:resource="&owl;Class" />
</rdf:Description>
```

---

```
<owl:Class rdf:about="#Man" />
```

- A string literal can be expressed as the object of an OWL statement or as XML attribute.

```
<myNs:Person rdf:about="#JohnDoe">
  <myNs:hasName>John</myNs:hasName>
  <myNs:hasSurname>Doe</myNs:hasSurname>
</myNs:Person>
```

---

```
<myNs:Person rdf:about="&myNs;JohnDoe"
  myNs:hasName="John" myNs:hasSurname="Doe" />
```

- Blank nodes are used to identify unnamed individuals. The following two OWL snippets identify the same resource.

```

<myNs:Person rdf:about="#John">
  <myNs:hasChild rdf:nodeID="node1" />
</myNs:Person>
<myNs:Child rdf:nodeID="node1">
  <myNs:hasName>Paul</myNs:hasName>
</myNs:Child>

```

---

```

<myNs:Person rdf:about="#John">
  <myNs:hasChild rdf:parseType="Resource">
    <rdf:type rdf:resource="#Child"/>
    <myNs:hasName>Paul</myNs:hasName>
  </myNs:hasChild>
</myNs:Person>

```

**Language identification attributes.** The language of a value can be defined with the *xml:lang* attribute in tags.

```

<owl:Class rdf:about="&myNs;Book">
  <rdfs:label xml:lang="en">Book</rdfs:label>
  <rdfs:label xml:lang="es">Libro</rdfs:label>
</owl:Class>

```

### 3.3 Description of the benchmarks

Each benchmark of the benchmark suite, as Table 3.1 shows, is described by an **identifier** unique, a **description** in natural language of the benchmark, a **formal description** in the Description Logics notation of the ontology, a **graphical representation** of the ontology, and a **file** with the ontology in the RDF/XML syntax<sup>6</sup>.

The OWL Lite Import Benchmark Suite is available in a public web page<sup>7</sup> and is composed of 82 benchmarks that are classified in 12 groups, each identified by one letter (from **A** to **L**). The list of all the benchmarks composing the benchmark suite can be found in Appendix A; the OWL files have not been included here, but they can be found in the benchmark suite web page.

Moreover, since OWL Lite has an underlying Description Logics semantics, we have also provided a description of all the benchmarks both in natural language and in Description Logics formalism. These descriptions can be found in Appendix B.

---

<sup>6</sup>All the files have been syntactically validated against the WonderWeb OWL Ontology Validator (<http://phoebus.cs.man.ac.uk:9999/OWL/Validator>)

<sup>7</sup><http://knowledgeweb.semanticweb.org/benchmarking interoperability/owl/import.html>

<b>Identifier</b>	<b>ISG03</b>
<b>Description</b>	Import a single functional object property whose domain is a class and whose range is another class
<b>Formal description</b>	$\top \sqsubseteq \leq 1 \text{ hasHusband}$ $\top \sqsubseteq \forall \text{hasHusband}^-. \text{Woman}$ $\top \sqsubseteq \forall \text{hasHusband}. \text{Man}$
<b>Graphical representation</b>	
<b>RDF/XML file</b>	<pre> ... &lt;owl:Class rdf:about="&amp;ex;Woman" /&gt; &lt;owl:Class rdf:about="&amp;ex;Man" /&gt; &lt;owl:ObjectProperty rdf:about="&amp;ex;hasHusband"&gt;   &lt;rdf:type rdf:resource="&amp;owl;FunctionalProperty" /&gt;   &lt;rdfs:domain rdf:resource="&amp;ex;Woman" /&gt;   &lt;rdfs:range rdf:resource="&amp;ex;Man" /&gt; &lt;/owl:ObjectProperty&gt; ... </pre>

Table 3.1: The description of a benchmark of the OWL Lite Import Benchmark Suite

### 3.4 Towards benchmark suites for OWL DL and OWL Full

Although the OWL Lite Import Benchmark Suite described in this chapter just deals with the OWL Lite sublanguage, it could also be used for evaluating the importers from OWL DL and OWL Full of Semantic Web tools.

However, the definition of the OWL Lite Import Benchmark Suite does not take into account the OWL vocabulary terms whose use is not allowed in OWL Lite. In addition, the use of the OWL vocabulary terms is restricted in both OWL Lite and OWL DL. Hence, the benchmark suite defined for OWL Lite is incomplete for OWL DL and OWL Full.

The next Sections analyze the possibility of extending the OWL Lite Import Benchmark Suite to cover OWL DL and OWL Full, examining the differences between the three species of OWL.

#### 3.4.1 OWL DL

As we mentioned above, it is not necessary to develop from scratch a new benchmark suite to evaluate the import of OWL DL ontologies; the OWL Lite Import Benchmark Suite can be extended by implementing an OWL DL Import Benchmark Suite on top of it.

As Figure 3.1 shows, to cover the OWL DL sublanguage of OWL, we should also need to consider:

- The different combinations of the OWL Lite vocabulary terms according to their

use in OWL DL, since OWL DL imposes fewer restrictions to their use. Table 3.2 shows the differences in the restrictions of use of the vocabulary terms for OWL Lite and DL<sup>8</sup>.

- The different combinations of the OWL DL vocabulary terms not allowed in OWL Lite, between themselves and between the OWL Lite vocabulary terms. The vocabulary terms allowed in OWL DL and not allowed in OWL Lite are: *owl:oneOf*, *owl:disjointWith*, *owl:unionOf*, *owl:complementOf*, *owl:hasValue*, and *owl:DataRange*.

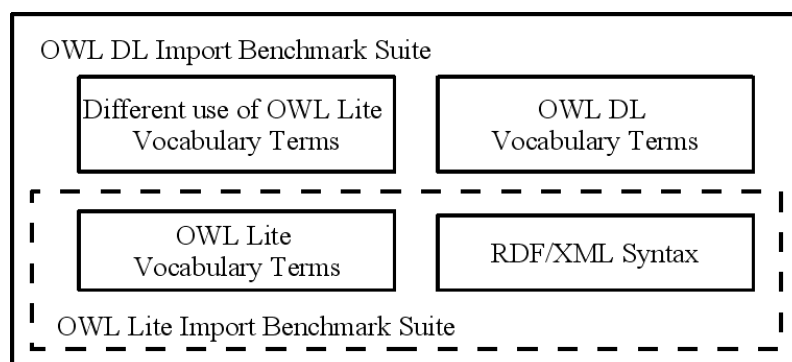


Figure 3.1: The OWL DL Import Benchmark Suite

For example, if we wanted to extend the benchmarks for *owl:equivalentClass* and *rdfs:subClassOf* we should define new benchmarks that consider as the subject and object of these properties all the different types of class descriptions allowed in OWL:

- A class identifier. These benchmarks are already defined for OWL Lite.
- An exhaustive enumeration of individuals. These benchmarks are not defined for OWL Lite.
- Property restrictions with value and cardinality constraints. Benchmarks are defined for OWL Lite considering restrictions in the object of the properties with 0 and 1 cardinality constraints. New benchmarks should be defined for cardinalities greater than 1 in the object of the properties and for restrictions in the subject of the properties.
- Set operators. Benchmarks are defined for OWL Lite by considering intersections in the object of the properties. New benchmarks should be defined for intersections in the subject of the properties and for union and complement in the subject and object of the properties.

<sup>8</sup><http://www.w3.org/TR/owl-ref/#Sublanguages-def>

Vocabulary Terms	OWL Lite restrictions	OWL DL restrictions
<i>owl:cardinality</i> <i>owl:minCardinality</i> <i>owl:maxCardinality</i>	Object must be 0 or 1	Object must be any integer $\geq 0$
<i>owl:equivalentClass</i> <i>rdfs:subClassOf</i>	Subject must be class names	No restriction
<i>owl:equivalentClass</i> <i>rdfs:subClassOf</i> <i>rdf:type</i>	Object must be class names or restrictions	No restriction
<i>rdfs:domain</i>	Object must be class names	No restriction
<i>owl:allValuesFrom</i> <i>owl:someValuesFrom</i> <i>rdfs:range</i>	Object must be class names or datatype names	No restriction
<i>owl:intersectionOf</i>	Used only with lists of class names or restrictions whose length is greater than 1	No restriction

Table 3.2: Restrictions in the use of OWL Lite and OWL DL

Following this approach, a considerable part of the benchmarks could be reused without any modification and, therefore, any tool that had already performed the experiments of the OWL Lite Import Benchmark Suite would not need to repeat them.

Nevertheless, when relaxing the restrictions of use of the OWL vocabulary terms from OWL Lite to OWL DL, a quite larger number of new benchmarks would be defined, which would affect the usability of the whole benchmark suite.

### 3.4.2 OWL Full

OWL Full has the same vocabulary terms as OWL DL, but it places no restrictions in their use. In fact, OWL Full is a superset of RDF(S), that gives the user the freedom to extend the RDF(S) vocabulary with the OWL constructors and to augment the meaning of both vocabularies.

The main characteristics of the use of OWL Full that are relevant to our case are:

- All the RDF(S) vocabulary can be used within OWL Full.
- OWL Full has no separation between classes, datatypes, datatype properties, object properties, annotation properties, individuals, data values, and the built-in vocabulary.
- Axioms in OWL Full do not have to be well formed.

This lack of restrictions implies that the use and possible combinations of the vocabulary terms in OWL DL and OWL Full is highly different. To develop a benchmark suite for evaluating the import of OWL Full ontologies, it might not be sufficient to develop some new benchmarks on top of the import benchmark suite for OWL DL, although it might be necessary to create a whole new benchmark suite that covers all the differences between OWL DL and OWL Full.

This import benchmark suite for OWL Full should consider all the possible combinations of the OWL and RDF(s) vocabularies terms and, because the number of these combinations is high, it would be necessary to prune the generation of benchmarks as it was done for the RDF(S) Import Benchmark Suite [García-Castro *et al.*, 2006].

# Chapter 4

## The IBSE tool

by RAÚL GARCÍA-CASTRO AND JESÚS PRIETO-GONZÁLEZ

IBSE (Interoperability Benchmark Suite Executor) is the evaluation infrastructure that automates the execution of the experiments of the OWL Interoperability Benchmarking. It offers a simple way of analysing the results, and permits smoothly including new tools into the infrastructure.

The source code and binaries of IBSE can be downloaded from its web page<sup>1</sup>. The latest version of the IBSE source code is located in a Subversion repository<sup>2</sup>.

This chapter starts by describing the requirements of the IBSE tool. Then, it presents some details of its implementation and of how to use it. Finally, it presents an example of the reports generated by IBSE.

### 4.1 IBSE requirements

The main requirements taken into account in the development of the IBSE tool are the following:

- **To be able to perform the experiments in as many tools as possible.** The OWL Interoperability Benchmarking considers any Semantic Web tool able to read and write ontologies from/to OWL files as a potential participant. Therefore, the IBSE tool should allow most of the existing tools to participate in the experiments (ontology repositories, ontology merging and alignment tools, reasoners, ontology-based annotation tools, etc.).
- **To automate the experiment execution and the analysis of the results.** In the OWL Interoperability Benchmarking we sacrifice a higher detail in results to avoid

---

<sup>1</sup>[http://knowledgeweb.semanticweb.org/benchmarking\\_interoperability/ibse/](http://knowledgeweb.semanticweb.org/benchmarking_interoperability/ibse/)

<sup>2</sup>[http://delicias.dia.fi.upm.es/repos/interoperability\\_benchmarking/](http://delicias.dia.fi.upm.es/repos/interoperability_benchmarking/)

that the humans conducted the experiments. However, full automation of the result analysis is not possible since this requires an individual to interpret them. Nevertheless, the evaluation infrastructure should automatically generate different visualizations and summaries of the results in different formats (such as HTML or SVG) to draw some conclusions at a glance. It is clear that an in-depth analysis of these results will still be needed for extracting the cause of the problems encountered and learning the improvement recommendations and the practices performed by developers.

- **To define benchmarks and results through ontologies.** The automation mentioned above requires that both, the benchmarks and the results, be machine-processable; therefore, we have represented them through ontologies. Instances of these ontologies will include the information needed to execute the benchmarks and the results obtained in their execution. This way of defining benchmarks and results allows having different predefined benchmark suites and execution results available in the Web, which can be used by anyone, for example, to classify and select tools according to their results, to execute the benchmarks in other tools or to process the accumulated results of different benchmark executions over time.
- **To use any group of ontologies as input for the experiments.** Executing benchmarks with no human effort can provide further advantages. The evaluation infrastructure should generate benchmark descriptions from any group of ontologies in RDF(S) or OWL, and should execute these benchmarks. Thus, different experiments could be easily performed with large numbers of ontologies, with domain-specific ontologies, with systematically-generated ontologies, etc.
- **To separate benchmark execution and report generation.** As a practical requirement, the evaluation infrastructure should be able to perform benchmark execution independently and to generate reports from one set of execution results, foreseeing experiment executions over a large number of tools, in different times, or by different parties.

## 4.2 IBSE implementation

The IBSE tool has been implemented using Java. It uses the *benchmarkOntology* and the *resultOntology*, respectively, to represent the benchmarks and the results that are presented in Section 4.2.1.

A normal execution of the IBSE tool comprises three consecutive steps, although they can also be executed independently. These steps are the following:

1. **To generate machine-readable benchmark descriptions from a group of ontologies.** In this step, a RDF file is generated; this file includes one benchmark



for each ontology of a group of ontologies located in a URI and the vocabulary of the *benchmarkOntology* ontology. This description generation can be skipped if benchmark descriptions are already available.

2. **To execute the benchmarks.** In this step, each benchmark described in the RDF file is executed interchanging between each pair of tools the ontology that it contains, being one tool the origin of the interchange and the other the destination of the interchange. The results are stored in a RDF file, employing the vocabulary of the *resultOntology* ontology.

Once we have the original, intermediate and final files with their corresponding ontologies, we extract the execution results by comparing each of these ontologies as shown in Section 2.2.1. This comparison and its output depend on an external ontology comparer. The current implementation uses the *diff* methods of a RDF(S) comparer (rdf-utils<sup>3</sup> version 0.3b) and of an OWL comparer (KAON2 OWL Tools<sup>4</sup> version 0.27). This implementation, however, permits inserting other comparers.

3. **To generate HTML files with different visualizations of the results.** In this step, different HTML files are generated with different visualizations, summaries and statistics of the results.

### 4.2.1 Representation of benchmarks and results

This section describes the two OWL ontologies employed in the IBSE tool: the *benchmarkOntology*<sup>5</sup> ontology, which defines the vocabulary that represents the benchmarks to be executed, and the *resultOntology*<sup>6</sup> ontology, which defines the vocabulary that represents the results of a benchmark execution.

These ontologies are lightweight since their main goal is to be user-friendly; they are described in Appendix C using the RDF/XML syntax.

Next, the section presents the classes and properties that these ontologies contain. All the datatype properties have as range *xsd:string* with the exception of *timestamp* whose range is *xsd:dateTime*.

**benchmarkOntology.** The *Document* class represents a document containing one ontology. A document can be further described by the following properties, which have *Document* as domain: *documentURL* (the URL of the document), *ontologyName*

---

<sup>3</sup><http://wymiwyg.org/rdf-utils/>

<sup>4</sup><http://owltools.ontoware.org/>

<sup>5</sup>[http://knowledgeweb.semanticweb.org/benchmarking\\_interoperability/owl/benchmarkOntology.owl](http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/benchmarkOntology.owl)

<sup>6</sup>[http://knowledgeweb.semanticweb.org/benchmarking\\_interoperability/owl/resultOntology.owl](http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/resultOntology.owl)

(the ontology name), *ontologyNamespace* (the ontology namespace), and *representationLanguage* (the language used to implemented the ontology).

The *Benchmark* class represents a benchmark to be executed. A benchmark can be further described with the following properties that have *Benchmark* as domain. Such properties are: *id* (the benchmark identifier); *usesDocument* (the document that contains one ontology used as input); *interchangeLanguage* (the interchange language used); *author* (the benchmark author); and *version* (the benchmark version number).

**resultOntology.** The *Tool* class represents a tool that has participated as origin or destination of an interchange in a benchmark. A tool can be further described with the following properties that have *Tool* as domain. These properties are: *toolName* (the tool name), and *toolVersion* (the tool version number).

The *Result* class represents a result of a benchmark execution. A result can be further described with the following properties that have *Result* as domain. These properties are: *ofBenchmark* (the benchmark to which the result corresponds); *originTool* (the tool origin of the interchange); *destinationTool* (the tool destination of the interchange); *execution*, *executionStep1*, *executionStep2* (if the whole interchange, the first and the second steps are carried out without any execution problem, respectively); *interchange*, *interchangeStep1*, *interchangeStep2* (if the ontology has been interchanged correctly from the original tool to the destination tool, in the first step, and in the second step with no addition or loss of information, respectively); *informationAdded*, *informationAddedStep1*, *informationAddedStep2* (the triples added in the whole interchange, in the first step, and in the second step, respectively); *informationRemoved*, *informationRemovedStep1*, *informationRemovedStep2* (the triples removed in the whole interchange, in the first step, and the second step, respectively); and finally, *timestamp* (the date and time when the benchmark is executed).

## 4.2.2 Inserting a new tool

As the experimentation requires no human intervention, we can only access tools through application programming interfaces (APIs) or through batch executions. There are other ways of executing an application automatically (i.e., Web Service executions) but these are not present in the current tools. Nevertheless, to adapt the IBSE tool for including other types of executions should be quite straightforward.

The only operation that a tool has to perform to participate in the experiment, as seen in Section 2.2.1, is to import an ontology from a file and to export the imported ontology into another file.

To insert a new tool in the evaluation infrastructure only one method from the *ToolManager* interface has to be implemented: *void ImportExport(String importFile, String*

*exportFile*, *String ontologyName*, *String namespace*, *String language*). This method receives as input parameters the following: the location of the file with the ontology to be imported; the location of the file where the exported ontology has to be written; the name of the ontology; the namespace of the ontology; and the representation language of the ontologies respectively.

This method has already been implemented for the tools participating in the benchmarking, which are: GATE, Jena, KAON2, the NeOn Toolkit, Protégé-Frames, Protégé-OWL, SemTalk, SWI-Prolog, and WebODE.

Most of these tools provide Java interfaces for performing the import and export operations. In the case of non-Java tools (SemTalk and SWI-Prolog), these operations were performed by executing their binaries using the *java.lang.Runtime* class.

As an example, the following lines show the implementation of the method for Jena:

```
public void ImportExport(String importFile, String exportFile, String ontologyName,
                        String namespace, String language) throws BadURIException{
    Model model = ModelFactory.createDefaultModel();
    Model model_out = ModelFactory.createDefaultModel();
    try {
        // Import
        FileInputStream inFile = new FileInputStream(importFile);
        model = model.read(importFile,null,null);
        inFile.close();
        // Export
        FileOutputStream outFile = new FileOutputStream(exportFile);
        String queryString = "DESCRIBE ?x WHERE {?x ?y ?z}";
        Query query = QueryFactory.create(queryString);
        QueryExecution gexec = QueryExecutionFactory.create(query, model);
        model_out = gexec.execDescribe();
        model_out.write(outFile);
        model_out.close();
        model_out.close();
    } catch (FileNotFoundException e) { e.printStackTrace(); }
    } catch (IOException e) { e.printStackTrace(); } }
```

### 4.2.3 Inserting and evaluating ontology comparers

We mentioned before that the IBSE tool uses external software for comparing the ontologies resulting from the experiment. IBSE currently uses the *diff* methods of a RDF(S) comparer (rdf-utils<sup>7</sup> version 0.3b) and of an OWL comparer (KAON2 OWL Tools<sup>8</sup> version 0.27).

Nevertheless, other ontology comparers can also be inserted into the IBSE tool by implementing a method from the *Comparer* interface: *int CompareFiles(String origin\_file, String compared\_file, String added\_file, String deleted\_file, String language)*. This method receives the following input parameters: the location of the two files to be compared; the location of the two files in which the inserted and removed triples will be stored; and the language in which the ontologies are written respectively.

<sup>7</sup><http://wymiwyg.org/rdf-utils/>

<sup>8</sup><http://owltools.ontoware.org/>

The software used for comparing ontologies could have execution problems when it compares two ontologies. Therefore, we need a previous evaluation of this software to ensure the validity of the benchmarking results.

The evaluation of the comparers, which consisted on detecting errors in them, was performed in two steps:

1. The interoperability experiment was carried out with the tools participating in the benchmarking whose knowledge model is the same as that of the interchange language. In theory, these tools should interchange all the ontologies correctly because no ontology translation is required for doing so. In this step, we analysed the cases where the interchanged ontology was different than the original one.
2. The interoperability experiment was carried out with all the tools participating in the benchmarking. In this step, we analysed the cases where the comparison of two ontologies caused an execution error in the comparer.

In the case of OWL, IBSE currently uses the KAON2 OWL Tools *diff* method for comparing OWL ontologies. The problems found in this ontology comparer after carrying out the previous steps were the following:

- When one of the ontologies is empty, the comparer returns that the ontologies are the same.
- The comparer returns complete definitions of the differences between the ontologies and not only the differing triples. For example, if two ontologies only differ in one triple:

```
Ontology 1:
ns1:Person rdfs:type owl:Class;
ns1:Person rdfs:label "Person";
```

```
Ontology 2:
ns1:Person rdfs:type owl:Class;
```

the comparer returns not just the triple but also the whole definition of the classes or properties involved:

```
Diff:
ns1:Person rdfs:type owl:Class
ns1:Person rdfs:label "Person";
```

- When the comparer compares two ontologies with blank nodes, it generates different node identifiers and, therefore, it returns that the ontologies are different.
- When one of the ontologies is not a valid OWL ontology in the RDF/XML syntax, the comparer throws an exception.

- The comparer is not robust and throws an exception when comparing ontologies with unexpected inputs, as for example, the incorrect class naming produced by some tools<sup>9</sup>, or the incorrect use of the OWL language constructors: use of *rdf:Property* instead of *owl:ObjectProperty* or *owl:DatatypeProperty*; use of a resource both as an object and as a datatype property; use of empty *rdfs:subClassOf* statements ("*<rdfs:subClassOf/>*"); or use of untyped object properties.

The first two problems were solved by adapting the output of the comparer inside IBSE. The behaviour of the ontology comparer in the rest of the cases was documented to be taken into account when analysing the interoperability results.

This is not an exhaustive evaluation of the comparer, but after analysing all the cases of the whole benchmarking results in which the interchanged ontologies were not the same, we found no more comparer errors.

### 4.3 Using IBSE

The only requirements for executing the evaluation infrastructure are to have a Java Runtime Environment and the IBSE binaries<sup>10</sup>. To perform the experiments with SemTalk and WebODE, these tools must also be installed in the system.

The steps to follow to perform the interoperability experiments using IBSE are the following:

1. To download the IBSE binaries.
2. To edit the *ibse.conf* file according to the user's execution preferences.
3. To prepare the tools wanted for the experiment. Some tools do not need any preparation as IBSE accesses them through their jars; others, however, do need preparation.
4. To run IBSE from the command line: *java -jar IBSE.jar [config file]*.

Steps 2. and 3. are optional for the default full execution of the experiments and for the generation of the reports. Nevertheless, the *ibse.conf* file allows customizing the execution by defining: a) the tools considered as the origin and the destination of the interchange (*ORIGIN\_TOOLS* and *DESTINATION\_TOOLS*); b) the language used in the input ontologies and in the interchange (*REPRESENTATION\_LANGUAGE*); c) the steps to perform in the execution (*DESCRIBE\_BENCHMARKS*, *EXECUTE\_BENCHMARKS*, *GENERATE\_REPORT\_FROM*); and d) the location of the data needed or generated by IBSE

---

<sup>9</sup>i.e., "*#http\_3A\_2F\_2Fwww.w3.org\_2F2002\_2F07\_2Fowl\_23Thing*"

<sup>10</sup><http://knowledgeweb.semanticweb.org/benchmarking-interoperability/ibse/files/IBSEv1.0.zip>

(*ONTOLOGIES\_URL*, *BENCHMARKS\_URL*, *RESULTS\_URL*, *RESULTS\_HTML\_URL*, *RESULTS\_RDF\_URL*). Full use details of use can be found in the comments of the *ibse.conf* file.

A future improvement of the IBSE tool could be its integration with testing infrastructures, such as JUnit.

After a full IBSE execution, the following files are generated in the results directory:

- One RDF file (*benchmarkDescriptions.rdf*) with the description of the benchmarks from the selected group of ontologies. The RDF file with the description of the benchmarks to be executed in the OWL Interoperability Benchmarking can be generated or downloaded from the Web<sup>11</sup>.
- RDF files (*Result<Tool1><Tool2>.rdf*) with the descriptions of the results for each pair of tools.
- The ontologies resulting from executing the experiments, the intermediate and final ones inclusive.
- The following HTML files with different visualizations, summaries and statistics of the results:
  - One index page to access all the reports.
  - Five pages for each combination of tools (both as origin and destination). One of the pages shows some statistics of the results; other shows the original, intermediate and final ontologies obtained in the benchmark executions; and the other three summarize the *Execution*, *Interchange*, *Information added*, and *Information lost* results contained in the RDF result files. These three pages show, for each benchmark, the results of the final interchange and of the intermediate steps (*Step 1* and *Step 2*), with different levels of detail.
  - For each pair of tools, one page summarizes the *Interchange* result considering one tool as origin and the other as destination of the interchange and vice versa.
  - For each tool, one page with the results of every benchmark execution, being this tool the origin and the other tools the destination of the interchange.

---

<sup>11</sup><http://knowledgeweb.semanticweb.org/benchmarking-interoperability/owl/OIBS.rdf>

## Chapter 5

# OWL interoperability results and analysis

by RAÚL GARCÍA-CASTRO

In this chapter we present the analysis of the interoperability using OWL as the interchange language of the Semantic Web tools that participated in the benchmarking.

For each of the tools, the analysis is divided in two consecutive steps (described in detail below):

1. To analyse the behaviour of the tool in the combined import and export operation.
2. To analyse the interoperability of the tool with all the tools participating in the benchmarking (including itself).

With the analyses, we provide references to the ontology or ontologies that originated the comment; their names appear in parentheses, i.e., (*ISA01-ISA03*).

### 5.1 Analysis of the import and export operation

Here we describe how the tool behaves in the combined operation of importing one OWL ontology and exporting it again (a step of the experiment, as defined in Section 2.2.1). To analyse the behaviour of the tool in one step of the experiment (a combined import and export operation), we have considered the tool results when such tool is the origin of the interchange (*Step 1*), irrespective of the tool that is the destination of the interchange. This step has as input an original ontology that is imported by the tool and then exported into a resultant ontology. This analysis has been performed by comparing the original and the resultant ontologies.

The results of a step execution in a tool can be classified into six categories:

- The original and the resultant ontologies are the same.
- The resultant ontology includes more information than the original one.
- The resultant ontology includes less information than the original one. In this case, information is sometimes inserted into the resultant ontology.
- The execution fails in the import and export operation.
- The execution fails when comparing the ontologies.

Table 5.1 presents a summary of the number of benchmarks for each tool<sup>1</sup>.

	<b>GA</b>	<b>JE</b>	<b>K2</b>	<b>PF</b>	<b>PO</b>	<b>ST</b>	<b>SP</b>	<b>WE</b>
<b>Same</b>	64	67	56		67	30	67	
<b>More</b>								8
<b>Less</b>	2		3	55		29		32
<b>Tool fails</b>								18
<b>Comparer fails</b>	1		8	12		8		9

Table 5.1: Summary of the results of the import and export operation

Below, we present the detailed results of each of the tools.

### 5.1.1 GATE results in the import and export operation

The different step executions usually produce the same ontology in GATE. In some cases, the execution of the comparer fails with an ontology generated by GATE (although the ontology validates correctly).

The results of a step execution in GATE, as shown in Figure 5.1, can be classified into three categories:

- The original and the resultant ontologies are the same. This occurs in 64 cases (ISA01-17, ISB01-12, ISC01-02, ISD01-04, ISE01-10, ISF01-03, ISG01-05, ISI01-05, ISJ01-03, ISK01-03).
- The resultant ontology includes less information than the original one. In this case, information is sometimes inserted into the resultant ontology. This occurs in 2 cases (ISH01, ISH03).
- The execution fails when comparing the ontologies. This occurs in 1 case (ISH02).



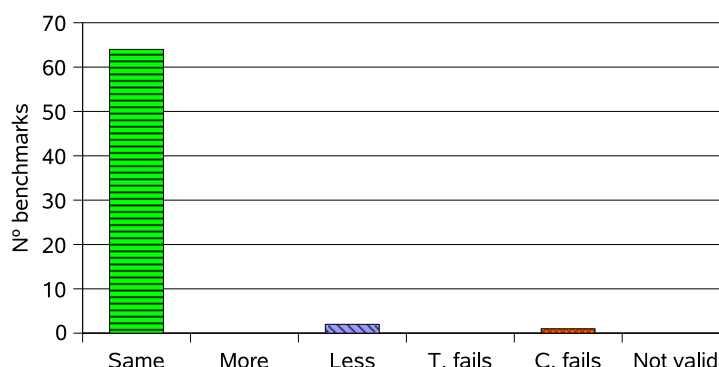


Figure 5.1: Results of the import and export operation for GATE

Next we describe the behaviour of GATE in one step, focusing on the combination of components present in the original ontology.

### Class hierarchies

- **Named class hierarchies without cycles (ISA01-04).** When a class is a subclass of several classes and of multiple classes that are a subclass of a class, one of the parent classes is not typed as a class. This converts the ontology into OWL Full.
- **Named class hierarchies with cycles (ISA05-06).** The ontologies processed remain the same.
- **Classes that are a subclass of a value constraint in an object property (ISA07-08).** The class defined inside the restriction is not typed as a class (OWL Full).
- **Classes that are a subclass of a cardinality constraint in an object or datatype property (ISA09-16).** The ontologies processed remain the same.
- **Classes that are a subclass of a class intersection (ISA17).** The ontologies processed remain the same.

### Class equivalences

- **Classes equivalent to named classes (ISB01).** The ontologies processed remain the same.
- **Classes equivalent to a value constraint in an object property (ISB02-03).** The class defined inside the restriction is not typed as a class (OWL Full).

<sup>1</sup>The tool names have been abbreviated in the table: GA=GATE, JE=Jena, K2=KAON2, PF=Protégé Frames, PO=Protégé OWL, ST=SemTalk, SP=SWI-Prolog, and WE=WebODE

- **Classes equivalent to a cardinality constraint in an object or datatype property (ISB04-11).** The ontologies processed remain the same.
- **Classes equivalent to a class intersection (ISB12).** The ontologies processed remain the same.

### Classes defined with set operators

- **Classes intersection of other classes (ISC01-02).** The ontologies processed remain the same.

### Properties

- **Object and datatype property hierarchies (ISD01-04).** The ontologies processed remain the same.
- **Object and datatype properties with or without domain or range, or with multiple domains or ranges (ISE01-10).** The ontologies processed remain the same.

### Relations between properties

- **Equivalent object and datatype properties (ISF01-02).** The ontologies processed remain the same.
- **Inverse object properties (ISF03).** The ontologies processed remain the same.

### Global cardinality constraints and logical property characteristics

- **Transitive, symmetric, or inverse functional object properties (ISG01-02,05).** The ontologies processed remain the same.
- **Functional object and datatype properties (ISG03-04).** The ontologies processed remain the same.

### Individuals

- **Individuals of a single class (ISH01,03).** One of the instances is lost.
- **Individuals of multiple classes (ISH02).** The comparer launches an exception but the ontologies processed remain the same.
- **Named individuals and object or datatype properties (ISI01-05).** The ontologies processed remain the same.

- **Anonymous individuals and object or datatype properties (ISJ01-03).** The result shows that the ontologies are different, but this is an error of the comparer. When the comparer compares two ontologies with blank nodes, it generates different node identifiers and, therefore, this implies that the ontologies are different.

### Individual identity

- **Equivalent or different individuals (ISK01-03).** The ontologies processed remain the same.

### 5.1.2 Jena results in the import and export operation

The different step executions do not produce any execution exception in Jena; in all the cases the original and the resultant ontologies are the same, as shown in Figure 5.2.

When there are anonymous individuals and object or datatype properties (ISJ01-03), the result shows that the ontologies are different, but this is an error of the comparer. When the comparer compares two ontologies with blank nodes, it generates different node identifiers and, therefore, it shows that the ontologies are different.

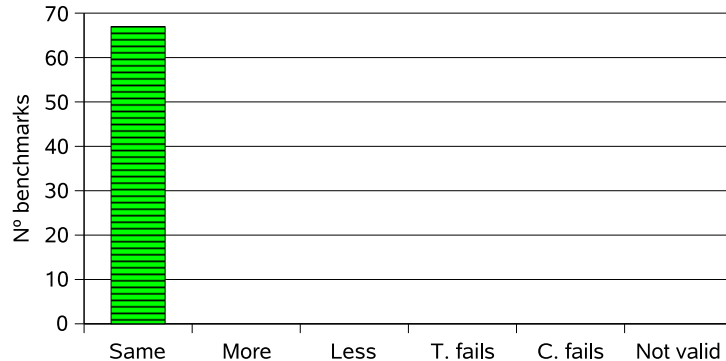


Figure 5.2: Results of the import and export operation for Jena

### 5.1.3 KAON2 results in the import and export operation

The different step executions usually produce the same ontology in KAON2. In some cases, the execution of the comparer fails with an ontology generated by KAON2 (although the ontology validates correctly).

The results of a step execution in KAON2, as shown in Figure 5.3, can be classified into three categories:

- The original and the resultant ontologies are the same. This occurs in 56 cases (ISA02-08, ISA10-11, ISA14-15, ISA17, ISB01-06, ISB09-10, ISB12, ISC01-02, ISD02, ISD04, ISE01-10, ISF01-03, ISG01-05, ISH01-03, ISI01-05, ISJ01-03, ISK01-03).
- The resultant ontology includes less information than the original one. In this case, information is sometimes inserted into the resultant ontology. This occurs in 3 cases (ISA01, ISD01, ISD03).
- The execution fails when comparing the ontologies. This occurs in 8 cases (ISA09, ISA12-13, ISA16, ISB04, ISB07-08, ISB11).

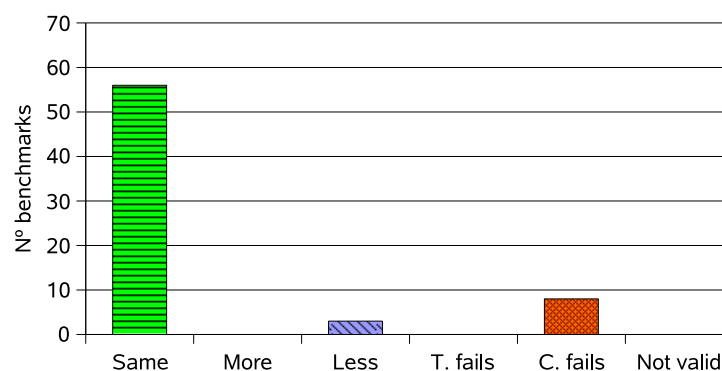


Figure 5.3: Results of the import and export operation for KAON2

Below, we describe the behaviour of KAON2 in one step, focusing on the combination of components present in the original ontology.

### Class hierarchies

- **A single class (ISA01).** The class is lost.
- **Named class hierarchies with or without cycles (ISA02-06).** The ontologies processed remain the same.
- **Classes that are a subclass of a value constraint in an object property (ISA07-08).**
- **Classes that are a subclass of an *owl:maxCardinality* or *owl:cardinality* cardinality constraint in an object or datatype property (ISA10-11,14-15).** The ontologies processed remain the same.
- **Classes that are a subclass of an *owl:minCardinality* cardinality constraint in an object or datatype property (ISA09,12-13,16).** The class is created as a subclass of a blank node instead of being created as a subclass of the restriction.

*rdfs:subClassOf* is used as a datatype property (OWL Full) and the class is considered an instance (*Individual(a:Employee value(rdfs:subClassOf ""))*)).

- **Classes that are a subclass of a class intersection (ISA17).** The ontologies processed remain the same.

### Class equivalences

- **Classes equivalent to named classes (ISB01).** The ontologies processed remain the same.
- **Classes equivalent to a value constraint in an object property (ISB02-03).** The ontologies processed remain the same.
- **Classes equivalent to an *owl:maxCardinality* or *owl:cardinality* cardinality constraint in an object or datatype property (ISB05-06,09-10).** The ontologies processed remain the same.
- **Classes equivalent to an *owl:minCardinality* cardinality constraint in an object or datatype property (ISB04,07-08,11).** The class is created as equivalent to a blank node instead of being created as equivalent to the restriction. *owl:equivalentClass* is used as a datatype property (OWL Full), and the class is considered an instance (*Individual(a:Employee value(owl:equivalentClass ""))*)).
- **Classes equivalent to a class intersection (ISB12).** The ontologies processed remain the same.

### Classes defined with set operators

- **Classes intersection of other classes (ISC01-02).** The ontologies processed remain the same.

### Properties

- **Object and datatype property hierarchies (ISD01-04).** When there is only one object or datatype property, the property is lost
- **Object and datatype properties with or without domain or range, or with multiple domains or ranges (ISE01-10).** The ontologies processed remain the same.

### Relations between properties

- **Equivalent object and datatype properties (ISF01-02).** The ontologies processed remain the same.
- **Inverse object properties (ISF03).** The ontologies processed remain the same.

### Global cardinality constraints and logical property characteristics

- **Transitive, symmetric, or inverse functional object properties (ISG01-02,05).** The ontologies processed remain the same.
- **Functional object and datatype properties (ISG03-04).** The ontologies processed remain the same.

### Individuals

- **Individuals of a single or multiple classes (ISH01-03).** The ontologies processed remain the same.
- **Named individuals and object or datatype properties (ISI01-05).** The ontologies processed remain the same.
- **Anonymous individuals and object or datatype properties (ISJ01-03).** The ontologies processed remain the same.

### Individual identity

- **Equivalent or different individuals (ISK01-03).** The ontologies processed remain the same.

## 5.1.4 Protégé-Frames results in the import and export operation

The different step executions never produce the same ontology in Protégé-Frames. However, with the ontologies generated by Protégé-Frames in some cases the execution of the comparer fails (although these ontologies validate correctly).

The results of a step execution in Protégé-Frames, as shown in Figure 5.4, can be classified into two categories:

- The resultant ontology includes less information than the original one. In this case, information is sometimes inserted into the resultant ontology. This occurs in 55 cases (ISA01-12, ISA17, ISB01-07, ISB12, ISC01-02, ISD01-04, ISE01-06, ISE08-10, ISF01-03, ISG01-05, ISH01-03, ISI01-03, ISJ01-02, ISK01-03).

- The execution fails when comparing the ontologies. This occurs in 12 cases (ISA13-16, ISB08-11, ISE07, ISI04-05, ISJ03).

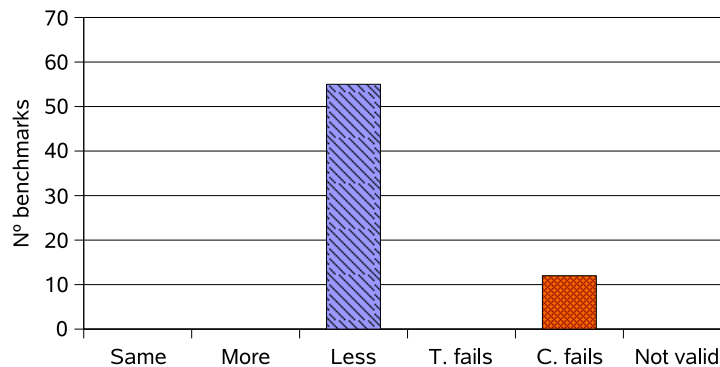


Figure 5.4: Results of the import and export operation for Protégé-Frames

Below, we describe the behaviour of Protégé-Frames in one step, focusing on the combination of components present in the original ontology.

### Ontologies

- The name of the ontology is changed into "*http://www.owl-ontologies.com/unnamed.owl*".

### Class hierarchies

- **Classes.** Class names are changed from "<class\_name>" to "ibs\_<class\_name>". A *rdfs:label* is inserted into the classes with the value "*ibs:<class\_name>^^xsd:string*". This occurs whenever classes appear.
- **Named class hierarchies without cycles (ISA01-04).** Classes are defined as a subclass of *owl:Thing*.
- **Named class hierarchies with cycles (ISA05-06).** When there are multiple classes, the classes are defined as equivalent.
- **Classes that are a subclass of a value constraint in an object property (ISA07-08).** Properties are created with a domain. In the case of the *owl:someValuesFrom* constraint, the constraint is lost. In the case of the *owl:allValuesFrom* constraint, classes are defined as a subclass of *owl:Thing*.
- **Classes that are a subclass of a cardinality constraint in an object property (ISA09-12).** Properties are created with a domain. In the case of the *owl:minCardinality* constraint, the constraint is lost. In the case of the *owl:maxCardinality* and *owl:cardinality* constraints, classes are defined as a subclass of *owl:Thing*.

- **Classes that are a subclass of a cardinality constraint in a datatype property (ISA13-16).** The datatype properties are changed into *rdf:Property*. Properties are created with a domain. In the case of the *owl:minCardinality* constraint, the constraint is lost. In the case of the *owl:maxCardinality* and *owl:cardinality* constraints, classes are defined as a subclass of *owl:Thing*. When *owl:maxCardinality* and *owl:minCardinality* constrain the same class, classes are defined as a subclass of *owl:Thing* and the domain of the property is defined as the union of the class.
- **Classes that are a subclass of a class intersection (ISA17).** The *owl:intersectionOf* property is lost but the ontologies are equivalent.

### Class equivalences

- **Classes equivalent to named classes (ISB01).** Classes are defined as a subclass of *owl:Thing*.
- **Classes equivalent to a value constraint in an object property (ISB02-03).** Properties are created with a domain. In the case of the *owl:someValuesFrom* constraint, the value constraint is lost. In the case of the *owl:allValuesFrom* value constraint, classes are defined as a subclass of *owl:Thing* and of the restriction instead of being defined as equivalent to the restriction.
- **Classes equivalent to a cardinality constraint in an object property (ISB04-07).** Properties are created with a domain. Classes are defined as a subclass of *owl:Thing*. Classes are defined as a subclass of the restriction instead of being defined as equivalent to the restriction. When *owl:maxCardinality* and *owl:minCardinality* constrain the same class, the domain of the property is defined as the union of the class.
- **Classes equivalent to a cardinality constraint in a datatype property (ISB08-11).** The datatype properties are changed into *rdf:Property*. Properties are created with a domain. Classes are defined as a subclass of the restriction instead of being defined as equivalent to the restriction. In the case of the *owl:minCardinality* constraint, the constraint is lost. In the case of the *owl:maxCardinality* and *owl:cardinality* constraints, classes are defined as a subclass of *owl:Thing*. When *owl:maxCardinality* and *owl:minCardinality* constrain the same class, classes are defined as a subclass of *owl:Thing*, and the domain of the property is defined as the union of the class.
- **Classes equivalent to a class intersection (ISB12).** The *owl:intersectionOf* property is lost. The classes of the intersection are defined as a subclass of the class.



### Classes defined with set operators

- **Classes intersection of other classes (ISC01-02).** The *owl:intersectionOf* property is lost. The classes of the intersection are defined as a subclass of the class.

### Properties

- **Object and datatype properties.** Property names are changed from "<property\_name>" to "ibs:<property\_name>". A *rdfs:label* is inserted into the properties with the value "'ibs:<name>'^^xsd:string". This occurs whenever properties appear. When there are object or datatype properties with range, the range is lost.
- **Object property hierarchies (ISD01-02).** The *rdfs:subPropertyOf* property is lost.
- **Datatype property hierarchies (ISD03-04).** The datatype properties are changed into *rdf:Property*. The *rdfs:subPropertyOf* property is lost.
- **Object properties with or without domain or range (ISE01-04).** No further issues have been identified besides those mentioned for object and datatype properties.
- **Object properties with multiple domains or ranges (ISE05-06).** When there are object properties with multiple domains, all domains except one are lost.
- **Datatype properties without domain or range (ISE07-08).** The datatype properties are changed into *rdf:Property*.
- **Datatype properties with domain and range (ISE09).** The datatype properties are changed into object properties.
- **Datatype properties with multiple domains (ISE10).** The datatype properties are changed into object properties. All domains except one are lost.

### Relations between properties

- **Equivalent object and datatype properties (ISF01-02).** The *owl:equivalentProperty* property is lost.
- **Inverse object properties (ISF03).** No further issues have been identified besides those mentioned for object and datatype properties.

### Global cardinality constraints and logical property characteristics

- **Transitive or symmetric object properties (ISG01-02).** The transitivity and the symmetry are lost.
- **Functional object and datatype properties (ISG03-04).** The datatype properties are changed into object properties.
- **Inverse functional object properties (ISG05).** The inverse functionality is lost.

### Individuals

- **Individuals.** The names of individuals are changed from "<individual\_name>" to "ibs\_<individual\_name>". A *rdfs:label* is inserted into the individuals with the value "'ibs:<name>'<sup>^</sup>xsd:string". This occurs whenever individuals appear.
- **Individuals of a single class (ISH01,03).** The individuals remain the same.
- **Individuals of multiple classes (ISH02).** All the type properties except one are lost.
- **Named individuals and object or datatype properties (ISI01-05).** When there are named individuals and datatype properties, the datatype properties are changed into object properties.
- **Anonymous individuals and object or datatype properties (ISJ01-03).** The anonymous individual is created as a named individual. When there are named individuals and datatype properties, the datatype properties are changed into object properties.

### Individual identity

- **Equivalent or different individuals (ISK01-03).** The properties and classes that define the equivalence or difference (*owl:sameAs*, *owl:different*, *owl:AllDifferent*) are lost.

## 5.1.5 Protégé-OWL results in the import and export operation

The different step executions do not produce any exception in Protégé-OWL; in all the cases, the original and the resultant ontologies are the same, as shown in Figure 5.5.

When there are anonymous individuals and object or datatype properties (ISJ01-03), the result shows that the ontologies are different, but this is an error of the comparer.

When the comparer compares two ontologies with blank nodes, it generates different node identifiers and, therefore, it shows that the ontologies are different.

On the other hand, when there are inverse object properties, the result shows that the ontologies are different, even though it is semantically the same. The only change is that Protégé-OWL defines the *owl:inverseOf* property in both properties instead of in just one.

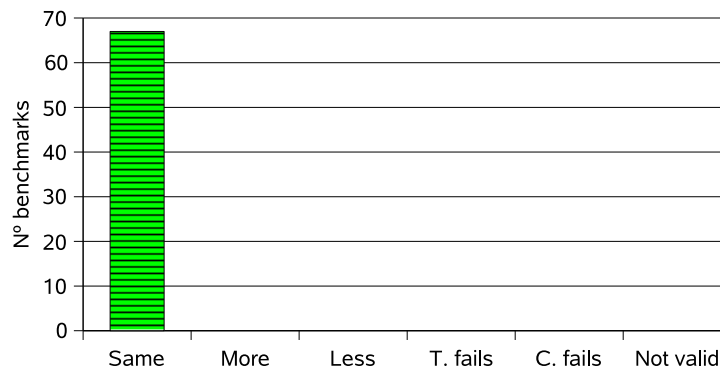


Figure 5.5: Results of the import and export operation for Protégé-OWL

### 5.1.6 SemTalk results in the import and export operation

The different step executions do not produce any execution exception in SemTalk; in some cases the execution of the comparer fails with the ontologies generated by SemTalk (although these ontologies validate correctly).

The results of a step execution in SemTalk, as shown in Figure 5.6, can be classified into three categories:

- The original and the resultant ontologies are the same. This occurs in 30 cases (ISA01-04, ISA07, ISA17, ISC01-02, ISD01-03, ISE01-07, ISF01, ISG01-03, ISH01-03, ISI01-03, ISK01-02).
- The resultant ontology includes less information than the original one. In this case, information is sometimes inserted into the resultant ontology. This occurs in 29 cases (ISA05-06, ISA08, ISA13-16, ISB01-03, ISB08-12, ISD04, ISE08-10, ISF02-04, ISG05, ISI04-05, ISJ01-03, ISK03).
- The execution fails when comparing the ontologies. This occurs in 8 cases (ISA09-12, ISB04-07).

Below, we describe the behaviour of SemTalk in one step, focusing on the combination of components present in the original ontology.

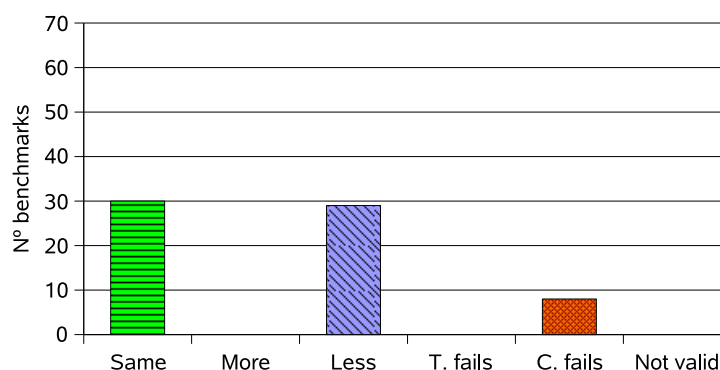


Figure 5.6: Results of the import and export operation for SemTalk

## Ontologies

- **Ontologies.** The name of the ontology is lost; it only appears in the *xml:ns* attribute as ontologies are created without the *rdf:about* attribute in the *owl:Ontology* statement (i.e., `<owl:Ontology />`). This occurs in all the ontologies.

## Class hierarchies

- **Named class hierarchies without cycles (ISA01-04).** The named class hierarchies remain the same.
- **Named class hierarchies with cycles (ISA05-06).** When there are cycles between multiple classes, one of the subclass properties is removed to avoid the cycle. When a class is a subclass of itself, the ontology processed is different but semantically the same. The statement that a class is a subclass of itself is removed.
- **Classes that are a subclass of a value constraint in an object property (ISA07-08).** In the case of the *owl:someValuesFrom* constraint, the subclass of the constraint remains the same. In the case of the *owl:allValuesFrom* constraint, the *owl:allValuesFrom* constraint is changed into *owl:someValuesFrom*.
- **Classes that are a subclass of a cardinality constraint in an object property (ISA09-12).** The object property is defined both as an object property and as a datatype property. The class is defined as a subclass of the restriction *restriction(a:hasName value ("^^xsd:string))*. In the case of the *owl:cardinality* constraint, the constraint is replaced by one *owl:minCardinality* constraint and one *owl:maxCardinality* constraint.
- **Classes that are a subclass of a cardinality constraint in a datatype property (ISA13-16).** The class is defined as a subclass of the restriction *restriction(a:hasName value ("^^xsd:string))*. In the case of the *owl:cardinality* constraint, the constraint

is replaced by one *owl:minCardinality* constraint and one *owl:maxCardinality* constraint.

- **Classes that are a subclass of a class intersection (ISA17).** The ontologies processed remain the same.

### Class equivalences

- **Classes equivalent to named classes (ISB01).** The *owl:equivalentClass* property is lost.
- **Classes equivalent to a value constraint in an object property (ISB02-03).** Classes are defined as a subclass instead of being defined as equivalent to the restriction. In the case of the *owl:someValuesFrom* constraint, the subclass of the constraint remains the same. In the case of the *owl:allValuesFrom* constraint, the constraint is changed into *owl:someValuesFrom*.
- **Classes equivalent to a cardinality constraint in an object property (ISB04-07).** Classes are defined as a subclass instead of being defined as equivalent to the restriction. The object property is defined both as an object property and as a datatype property. The class is defined as a subclass of the restriction *restriction(a:hasName value ("^^xsd:string))*. In the case of the *owl:cardinality* constraint, the constraint is replaced by one *owl:minCardinality* constraint and one *owl:maxCardinality* constraint.
- **Classes equivalent to a cardinality constraint in a datatype property (ISB08-11).** Classes are defined as a subclass instead of being defined as equivalent to the restriction. In the case of the *owl:cardinality* constraint, the constraint is replaced by one *owl:minCardinality* constraint and one *owl:maxCardinality* constraint.
- **Classes equivalent to a class intersection (ISB12).** The *owl:intersectionOf* property is lost.

### Classes defined with set operators

- **Classes intersection of other classes (ISC01-02).** The ontologies processed remain the same.

### Properties

- **Object and datatype property hierarchies (ISD01-04).** The ontologies processed remain the same.

- **Object properties with or without domain or range or with multiple domains and ranges (ISE01-06).** The ontologies processed remain the same.
- **Datatype properties with or without domain or range or with multiple domains (ISE07-10).** The range is lost.

### Relations between properties

- **Equivalent object and datatype properties (ISF01-02).** When there are datatype properties, the range is lost.
- **Inverse object properties (ISF03).** The *owl:inverseOf* property is lost.

### Global cardinality constraints and logical property characteristics

- **Transitive or symmetric object properties (ISG01-02).** The ontologies processed remain the same.
- **Functional object and datatype properties (ISG03-04).** When there are datatype properties, the range is lost and also lost is the statement about the property being functional.
- **Inverse functional object properties (ISG05).** The statement about the property being inverse functional is lost.

### Individuals

- **Individuals of a single or multiple classes (ISH01-03).** The ontologies processed remain the same.
- **Named individuals and object or datatype properties (ISI01-05).** When there are datatype properties, the range is lost.
- **Anonymous individuals and object or datatype properties (ISJ01-03).** The anonymous individual is lost.

### Individual identity

- **Equivalent or different individuals (ISK01-03).** The *owl:sameAs* and *owl:different* properties are lost. In the case of the (*owl:AllDifferent*) class, the individuals are also instances of *owl:Thing*, even though it is semantically the same.

### 5.1.7 SWI-Prolog results in the import and export operation

The different step executions do not produce any execution exception in SWI-Prolog; in all the cases the original and the resultant ontologies are the same, as shown in Figure 5.7.

When there are anonymous individuals and object or datatype properties (ISJ01-03), the result shows that the ontologies are different, but this is an error of the comparer. When the comparer compares two ontologies with blank nodes, it generates different node identifiers and, therefore, it shows that the ontologies are different.

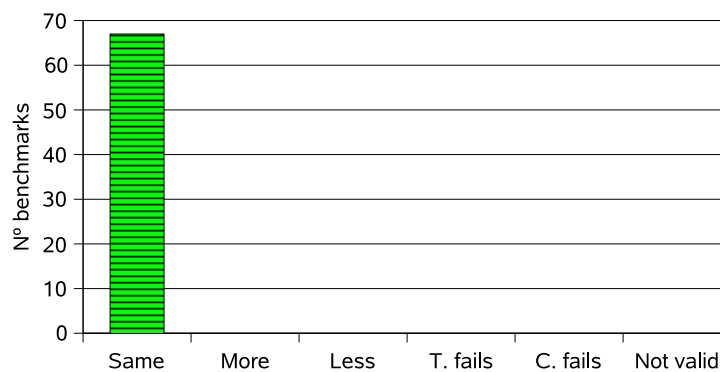


Figure 5.7: Results of the import and export operation for SWI-Prolog

### 5.1.8 WebODE results in the import and export operation

The different step executions never produce the same ontology in WebODE. However, in some cases, WebODE's execution fails, whereas in others, it is the execution of the comparer that fails with the ontologies generated by WebODE (although these ontologies validate correctly).

The results of a step execution in WebODE, as shown in Figure 5.8, can be classified into four categories:

- The resultant ontology includes more information than the original one. This occurs in 8 cases (ISA01, ISA08, ISD01, ISE02-04, ISE07-08).
- The resultant ontology includes less information than the original one. In this case, information is sometimes inserted into the resultant ontology. This occurs in 32 cases (ISA06-07, ISB01-03, ISB12, ISC01-02, ISD02-04, ISE09, ISF01-03, ISG01-04, ISH01, ISH03, ISI01-05, ISJ01-03, ISK01-03).
- The execution fails in the import and export operation. This occurs in 18 cases (ISA02-05, ISA13-17, ISB08-11, ISE05-06, ISE10, ISG05, ISH02).

- The execution fails when comparing the ontologies. This occurs in 9 cases (ISA09-12, ISB04-07, ISE01).

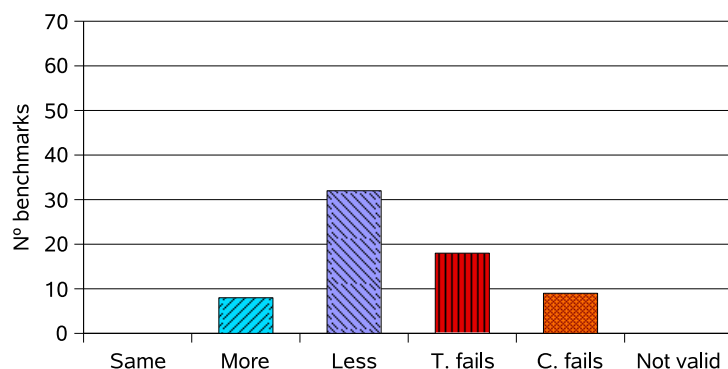


Figure 5.8: Results of the import and export operation for WebODE

Below, we describe the behaviour of WebODE in one step, focusing on the combination of components present in the original ontology.

### Class hierarchies

- **Classes.** A *rdfs:label* is inserted into the classes with the value "*<class\_name>*". This occurs whenever classes appear.
- **Named class hierarchies with or without cycles (ISA01-06).** When a hierarchy has multiple classes, execution fails. When a class is a subclass of itself, the ontology processed is different but semantically the same. It is only removed the statement about a class being a subclass of itself.
- **Classes that are a subclass of a value constraint in an object property (ISA07-08).** A new property is created with a name "*<property\_name>\_I*" and with an incorrect domain and range<sup>2</sup>. The restriction is created with the value constraint *owl:allValuesFrom(owl:Thing)*. In the case of the *owl:someValuesFrom* constraint, the constraint is lost.
- **Classes that are a subclass of a cardinality constraint in an object property (ISA09-12).** The property is created with a domain that is defined as the union of the class and an incorrect name<sup>2</sup>. The property is created with a range that is defined as the union of *owl:Thing* and an incorrect name<sup>2</sup>. The restriction is created on *owl:Thing* instead of on the property; therefore, *owl:Thing* is defined as an object property. The restriction is created with the *owl:allValuesFrom(owl:Thing)* value constraint. In the case of the *owl:minCardinality* constraint, the constraint in the

<sup>2</sup><http://www.w3.org/2002/07/owl#Thing>



restriction is lost. In the case of the *owl:maxCardinality* constraint, the value of the constraint is "11" instead of "1". In the case of the *owl:cardinality* constraint, the constraint is created as *owl:maxCardinality* instead of as *owl:cardinality* and the value of the constraint is "11" instead of "1".

- **Classes that are a subclass of a cardinality constraint in a datatype property (ISA13-16).** The execution fails.
- **Classes that are a subclass of a class intersection (ISA17).** The execution fails.

### Class equivalences

- **Classes equivalent to named classes (ISB01).** The *owl:equivalentClass* property is lost.
- **Classes equivalent to a value constraint in an object property (ISB02-03).** The property is created with domain and range, being the domain an anonymous concept and not the class. A new property is created with name "<property\_name>\_I" and with incorrect domain and range<sup>2</sup>. The anonymous concept is created as a subclass of the restriction and not as equivalent to the restriction. The restriction is created with the *owl:allValuesFrom(owl:Thing)* value constraint. In the case of the *owl:someValuesFrom* constraint, the constraint is lost.
- **Classes equivalent to a cardinality constraint in an object property (ISB04-07).** The property is created with a domain that is defined as the union of an anonymous concept and an incorrect name<sup>2</sup>. The property is created with a range that is defined as the union of *owl:Thing* and an incorrect name<sup>2</sup>. The anonymous concept is created as a subclass of the restriction and not as equivalent to the restriction. The restriction is created on *owl:Thing* instead of on the property, therefore, *owl:Thing* is defined as an object property. The restriction is created with the value constraint *owl:allValuesFrom(owl:Thing)*. In the case of the *owl:minCardinality* constraint, the constraint in the restriction is lost. In the case of the *owl:maxCardinality* constraint, the value of the constraint is "11" instead of "1". In the case of the *owl:cardinality* constraint, the constraint is created as *owl:maxCardinality* instead of as *owl:cardinality* and the value of the constraint is "11" instead of "1".
- **Classes equivalent to a cardinality constraint in a datatype property (ISB08-11).** The execution fails.
- **Classes equivalent to a class intersection (ISB12).** The *owl:intersectionOf* and *owl:equivalentClass* properties are lost. An anonymous class is created.

### Classes defined with set operators

- **Classes intersection of other classes (ISC01-02).** The *owl:intersectionOf* property is lost.

### Properties

- **Object and datatype properties.** A *rdfs:label* is inserted into the properties with the value "<property\_name>". This occurs whenever properties appear.
- **Object and datatype property hierarchies (ISD01-04).** The *rdfs:subPropertyOf* properties are lost.
- **Object properties without domain or range (ISE01-02).** When there are object properties without domain, the domain is created with an incorrect name<sup>2</sup>. When there are object properties without range, the range is created with an incorrect name<sup>2</sup> and the class is created as a subclass of the restriction *restriction(owl:Thing owl:allValuesFrom(owl:Thing))*.
- **Datatype properties without domain or range (ISE07-08).** When there are datatype properties without domain, the datatype property is lost. When there are datatype properties without range, the class is created as a subclass of the restriction *restriction(a:hasSSN owl:allValuesFrom(xsd:string))* and the range is created as *xsd:string*.
- **Object properties with domain and range (ISE03-04).** The class is created as a subclass of the restriction *restriction(a:hasChild owl:allValuesFrom(a:Person))*.
- **Object properties with domain and range (ISE09).** The class is created as a subclass of the restriction *restriction(a:hasSSN owl:allValuesFrom(xsd:string))*. The range changes from *rdfs:Literal* to *xsd:string*.
- **Object and datatype properties with multiple domains or ranges (ISE07-08,10).** The execution fails.

### Relations between properties

- **Equivalent object and datatype properties (ISF01-02).** The *owl:equivalentProperty* property is lost.
- **Inverse object properties (ISF03).** The *owl:inverseOf* property is lost.

### Global cardinality constraints and logical property characteristics

- **Transitive or symmetric object properties (ISG01-02).** The transitivity and the symmetry are lost.
- **Functional object and datatype properties (ISG03-04).** The class is created as a subclass of the restriction *restriction(a:hasHusband maxCardinality(1))*.
- **Inverse functional object properties (ISG05).** The execution fails.

### Individuals

- **Individuals.** A *rdfs:label* property is inserted into the individuals with the value "*<individual\_name>*". This occurs whenever individuals appear.
- **Individuals of a single class (ISH01,03).** The individuals remain the same.
- **Individuals of multiple classes (ISH02).** The execution fails.
- **Named individuals and object properties (ISI01-03).** The property with the value in the instance is lost.
- **Named individuals and datatype properties (ISI04-05).** The value in the property is changed from "*<value>*" to "*<value>*"<sup>^^xsd:string</sup>.
- **Anonymous individuals and object or datatype properties (ISJ01-03).** The anonymous individual is created as a named individual.

### Individual identity

- **Equivalent or different individuals (ISK01-03).** The properties and classes that define the equivalence or difference (*owl:sameAs*, *owl:different*, *owl:AllDifferent*) are lost.

## 5.2 Analysis of the interoperability

With the previous information about the behaviour of the tool in a step of the experiment, we provide the analysis of its interoperability with all the tools participating in the benchmarking (including itself). For performing such analysis, we have considered the results of its interoperability when this is the origin and the destination of the interchange with the other tools.

First, we present a table summarizing the results of the interoperability for each tool. Then, we present some interoperability issues not detected in the analysis of the import

and export operation. Finally, we highlight the ontology components that the tools are able to interchange.

### 5.2.1 Summary of the interoperability results

In the tables below<sup>3</sup>, the results of the interoperability between two tools (i.e., T1 and T2) have been grouped into categories, as in the previous section; the results also include the interchange from one tool to another (from T1 to T2) and vice versa (from T2 to T1). The results of the table are restrictive, i.e., when a single benchmark in a category has any problem in one of the directions of the interchange, the whole category states to have this problem. The results for any category can be the following:

- **SAME.** When all the ontologies interchanged between two tools are the same (all the benchmarks in the category have an *INTEROPERABILITY* result of *SAME*).
- **DIFF.** When at least one ontology interchanged between two tools is different and no execution errors exist (any benchmark in the category has an *INTEROPERABILITY* result of *DIFFERENT* and no benchmark with an *EXECUTION* result of *N.E.* exists).
- **N.E.** When at least one ontology could not be interchanged between two tools because of an execution error (any benchmark in the category has an *EXECUTION* result of *N.E.* - Non Executed).

Tables 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, and 5.9 show a summary of the results of the interoperability of GATE, Jena, KAON2, Protégé-Frames, Protégé-OWL, SemTalk, SWI-Prolog, and WebODE with the other tools, respectively.

---

<sup>3</sup>Tool names have been shortened in the table: GA=GATE, JE=Jena, K2=KAON2, PF=Protégé Frames, PO=Protégé OWL, ST=SemTalk, SP=SWI-Prolog, and WE=WebODE

Categories	Benchmarks	GA-GA	JE-GA	K2-GA	PF-GA	PO-GA	SP-GA	ST-GA	WE-GA
<b>Class hierarchies</b>									
Named class hierarchies without cycles	ISA01-ISA04	N.E.	SAME	N.E.	DIFF	N.E.	N.E.	N.E.	N.E.
Named class hierarchies with cycles	ISA05-ISA06	N.E.	SAME	N.E.	DIFF	N.E.	N.E.	DIFF	N.E.
Classes subclass of a value constraint in an object property	ISA07-ISA08	N.E.	SAME	N.E.	DIFF	N.E.	N.E.	N.E.	N.E.
Classes subclass of a cardinality constraint in an object property	ISA09-ISA12	N.E.	N.E.	N.E.	DIFF	N.E.	N.E.	N.E.	N.E.
Classes subclass of a cardinality constraint in a datatype property	ISA13-ISA16	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Classes subclass of a class intersection	ISA17	N.E.	SAME	SAME	DIFF	N.E.	SAME	N.E.	N.E.
<b>Class equivalences</b>									
Equivalent named classes	ISB01	SAME	SAME	N.E.	DIFF	SAME	SAME	DIFF	N.E.
Classes equivalent to a value constraint in an object property	ISB02-ISB03	N.E.	SAME	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Classes equivalent to a cardinality constraint in an object property	ISB04-ISB07	N.E.	N.E.	N.E.	DIFF	N.E.	N.E.	N.E.	N.E.
Classes equivalent to a cardinality constraint in a datatype property	ISB08-ISB11	N.E.	SAME	N.E.	N.E.	SAME	N.E.	N.E.	N.E.
Classes equivalent to a class intersection	ISB12	SAME	SAME	SAME	DIFF	SAME	N.E.	N.E.	N.E.
<b>Classes defined with set operators</b>									
Classes intersection of other classes	ISC01-ISC02	N.E.	SAME	N.E.	DIFF	SAME	N.E.	N.E.	N.E.
<b>Property hierarchies</b>									
Object property hierarchies	ISD01-ISD02	N.E.	DIFF	N.E.	DIFF	DIFF	N.E.	N.E.	N.E.
Datatype property hierarchies	ISD03-ISD04	DIFF	DIFF	N.E.	DIFF	N.E.	N.E.	DIFF	N.E.
<b>Properties with domain and range</b>									
Object properties without domain or range	ISE01-ISE02	N.E.	SAME	N.E.	DIFF	N.E.	N.E.	N.E.	N.E.
Object properties with domain and range	ISE03-ISE04	N.E.	SAME	N.E.	DIFF	N.E.	N.E.	N.E.	N.E.
Object properties with multiple domains or ranges	ISE05-ISE06	N.E.	N.E.	N.E.	DIFF	N.E.	SAME	N.E.	N.E.
Datatype properties without domain or range	ISE07-ISE08	N.E.	SAME	N.E.	N.E.	N.E.	N.E.	DIFF	N.E.
Datatype properties with domain and range	ISE09	N.E.	SAME	SAME	DIFF	N.E.	SAME	N.E.	N.E.
Datatype properties with multiple domains	ISE10	SAME	SAME	SAME	DIFF	N.E.	SAME	DIFF	N.E.
<b>Relations between properties</b>									
Equivalent object and datatype properties	ISF01-ISF02	N.E.	SAME	N.E.	DIFF	N.E.	SAME	N.E.	N.E.
Inverse object properties	ISF03	N.E.	SAME	N.E.	DIFF	N.E.	SAME	N.E.	N.E.
<b>Global cardinality constraints &amp; logical property characteristics</b>									
Transitive object properties	ISG01	SAME	SAME	SAME	DIFF	N.E.	N.E.	DIFF	N.E.
Symmetric object properties	ISG02	N.E.	SAME	N.E.	DIFF	SAME	SAME	N.E.	N.E.
Functional object and datatype properties	ISG03-ISG04	N.E.	SAME	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Inverse functional object properties	ISG05	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	N.E.
<b>Individuals</b>									
Instances	ISH01, ISH03	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Instances of multiple classes	ISH02	DIFF	DIFF	N.E.	N.E.	DIFF	N.E.	N.E.	N.E.
Named individuals and object properties	ISI01-ISI03	N.E.	SAME	N.E.	DIFF	N.E.	N.E.	N.E.	N.E.
Named individuals and datatype properties	ISI04-ISI05	N.E.	SAME	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Anonymous individuals and object properties	ISJ01-ISJ02	N.E.	N.E.	N.E.	DIFF	N.E.	N.E.	N.E.	N.E.
Anonymous individuals and datatype properties	ISJ03	DIFF	DIFF	DIFF	N.E.	N.E.	N.E.	N.E.	N.E.
<b>Individual identity</b>									
Equivalent individuals	ISK01	N.E.	N.E.	N.E.	DIFF	SAME	SAME	DIFF	N.E.
Different individuals	ISK02-ISK03	N.E.	DIFF	N.E.	N.E.	SAME	N.E.	N.E.	N.E.

Table 5.2: Summary of the results of the OWL interoperability of GATE

Categories	Benchmarks	GA-JE	JE-JE	K2-JE	PF-JE	PO-JE	SP-JE	ST-JE	WE-JE
<b>Class hierarchies</b>									
Named class hierarchies without cycles	ISA01-ISA04	SAME	SAME	DIFF	DIFF	SAME	SAME	DIFF	N.E.
Named class hierarchies with cycles	ISA05-ISA06	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	N.E.
Classes subclass of a value constraint in an object property	ISA07-ISA08	SAME	SAME	SAME	DIFF	SAME	SAME	N.E.	DIFF
Classes subclass of a cardinality constraint in an object property	ISA09-ISA12	N.E.	SAME	N.E.	DIFF	SAME	SAME	N.E.	N.E.
Classes subclass of a cardinality constraint in a datatype property	ISA13-ISA16	N.E.	SAME	N.E.	N.E.	SAME	SAME	N.E.	N.E.
Classes subclass of a class intersection	ISA17	SAME	SAME	SAME	DIFF	SAME	SAME	N.E.	N.E.
<b>Class equivalences</b>									
Equivalent named classes	ISB01	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Classes equivalent to a value constraint in an object property	ISB02-ISB03	SAME	SAME	SAME	DIFF	SAME	SAME	N.E.	DIFF
Classes equivalent to a cardinality constraint in an object property	ISB04-ISB07	N.E.	SAME	N.E.	DIFF	SAME	SAME	N.E.	N.E.
Classes equivalent to a cardinality constraint in a datatype property	ISB08-ISB11	SAME	SAME	N.E.	N.E.	SAME	SAME	N.E.	N.E.
Classes equivalent to a class intersection	ISB12	SAME	SAME	SAME	DIFF	SAME	SAME	N.E.	DIFF
<b>Classes defined with set operators</b>									
Classes intersection of other classes	ISC01-ISC02	SAME	SAME	SAME	DIFF	SAME	SAME	N.E.	DIFF
<b>Property hierarchies</b>									
Object property hierarchies	ISD01-ISD02	DIFF	SAME	DIFF	DIFF	SAME	SAME	DIFF	DIFF
Datatype property hierarchies	ISD03-ISD04	DIFF	SAME	DIFF	DIFF	SAME	SAME	DIFF	DIFF
<b>Properties with domain and range</b>									
Object properties without domain or range	ISE01-ISE02	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	N.E.
Object properties with domain and range	ISE03-ISE04	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Object properties with multiple domains or ranges	ISE05-ISE06	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	N.E.
Datatype properties without domain or range	ISE07-ISE08	SAME	SAME	SAME	N.E.	SAME	SAME	DIFF	DIFF
Datatype properties with domain and range	ISE09	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Datatype properties with multiple domains	ISE10	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	N.E.
<b>Relations between properties</b>									
Equivalent object and datatype properties	ISF01-ISF02	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Inverse object properties	ISF03	SAME	SAME	SAME	DIFF	DIFF	SAME	DIFF	DIFF
<b>Global cardinality constraints &amp; logical property characteristics</b>									
Transitive object properties	ISG01	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Symmetric object properties	ISG02	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Functional object and datatype properties	ISG03-ISG04	SAME	SAME	SAME	DIFF	SAME	SAME	N.E.	DIFF
Inverse functional object properties	ISG05	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	N.E.
<b>Individuals</b>									
Instances	ISH01, ISH03	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	N.E.
Instances of multiple classes	ISH02	DIFF	SAME	SAME	DIFF	SAME	SAME	SAME	DIFF
Named individuals and object properties	ISI01-ISI03	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Named individuals and datatype properties	ISI04-ISI05	SAME	SAME	SAME	N.E.	SAME	SAME	DIFF	DIFF
Anonymous individuals and object properties	ISJ01-ISJ02	N.E.	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF
Anonymous individuals and datatype properties	ISJ03	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF	N.E.	DIFF
<b>Individual identity</b>									
Equivalent individuals	ISK01	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Different individuals	ISK02-ISK03	DIFF	SAME	SAME	DIFF	SAME	SAME	N.E.	DIFF

Table 5.3: Summary of the results of the OWL interoperability of Jena

Categories	Benchmarks	GA-K2	JE-K2	K2-K2	PF-K2	PO-K2	SP-K2	ST-K2	WE-K2
<b>Class hierarchies</b>									
Named class hierarchies without cycles	ISA01-ISA04	N.E.	DIFF	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.
Named class hierarchies with cycles	ISA05-ISA06	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	N.E.
Classes subclass of a value constraint in an object property	ISA07-ISA08	N.E.	SAME	SAME	DIFF	SAME	SAME	N.E.	DIFF
Classes subclass of a cardinality constraint in an object property	ISA09-ISA12	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Classes subclass of a cardinality constraint in a datatype property	ISA13-ISA16	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Classes subclass of a class intersection	ISA17	SAME	SAME	SAME	DIFF	SAME	SAME	N.E.	N.E.
<b>Class equivalences</b>									
Equivalent named classes	ISB01	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Classes equivalent to a value constraint in an object property	ISB02-ISB03	N.E.	SAME	SAME	DIFF	SAME	SAME	N.E.	DIFF
Classes equivalent to a cardinality constraint in an object property	ISB04-ISB07	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Classes equivalent to a cardinality constraint in a datatype property	ISB08-ISB11	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Classes equivalent to a class intersection	ISB12	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
<b>Classes defined with set operators</b>									
Classes intersection of other classes	ISC01-ISC02	N.E.	SAME	SAME	DIFF	SAME	SAME	N.E.	DIFF
<b>Property hierarchies</b>									
Object property hierarchies	ISD01-ISD02	N.E.	DIFF	DIFF	DIFF	DIFF	DIFF	DIFF	DIFF
Datatype property hierarchies	ISD03-ISD04	N.E.	DIFF	DIFF	DIFF	DIFF	DIFF	DIFF	DIFF
<b>Properties with domain and range</b>									
Object properties without domain or range	ISE01-ISE02	N.E.	SAME	SAME	DIFF	SAME	SAME	SAME	N.E.
Object properties with domain and range	ISE03-ISE04	N.E.	SAME	SAME	DIFF	SAME	SAME	SAME	DIFF
Object properties with multiple domains or ranges	ISE05-ISE06	N.E.	SAME	SAME	DIFF	SAME	SAME	SAME	N.E.
Datatype properties without domain or range	ISE07-ISE08	N.E.	SAME	SAME	N.E.	SAME	SAME	DIFF	DIFF
Datatype properties with domain and range	ISE09	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Datatype properties with multiple domains	ISE10	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	N.E.
<b>Relations between properties</b>									
Equivalent object and datatype properties	ISF01-ISF02	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Inverse object properties	ISF03	N.E.	SAME	SAME	DIFF	DIFF	SAME	DIFF	DIFF
<b>Global cardinality constraints &amp; logical property characteristics</b>									
Transitive object properties	ISG01	SAME	SAME	SAME	DIFF	SAME	SAME	SAME	DIFF
Symmetric object properties	ISG02	N.E.	SAME	SAME	DIFF	SAME	SAME	SAME	DIFF
Functional object and datatype properties	ISG03-ISG04	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Inverse functional object properties	ISG05	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	N.E.
<b>Individuals</b>									
Instances	ISH01, ISH03	N.E.	SAME	SAME	DIFF	SAME	SAME	SAME	DIFF
Instances of multiple classes	ISH02	N.E.	SAME	SAME	DIFF	SAME	SAME	SAME	DIFF
Named individuals and object properties	ISI01-ISI03	N.E.	SAME	SAME	DIFF	SAME	SAME	SAME	DIFF
Named individuals and datatype properties	ISI04-ISI05	N.E.	SAME	SAME	N.E.	SAME	SAME	DIFF	DIFF
Anonymous individuals and object properties	ISJ01-ISJ02	N.E.	DIFF	SAME	DIFF	DIFF	DIFF	DIFF	DIFF
Anonymous individuals and datatype properties	ISJ03	DIFF	DIFF	SAME	N.E.	DIFF	DIFF	DIFF	DIFF
<b>Individual identity</b>									
Equivalent individuals	ISK01	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Different individuals	ISK02-ISK03	N.E.	SAME	SAME	DIFF	SAME	SAME	N.E.	N.E.

Table 5.4: Summary of the results of the OWL interoperability of KAON2

Categories	Benchmarks	GA-PF	JE-PF	K2-PF	PF-PF	PO-PF	SP-PF	ST-PF	WE-PF
<b>Class hierarchies</b>									
Named class hierarchies without cycles	ISA01-ISA04	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	N.E.
Named class hierarchies with cycles	ISA05-ISA06	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	N.E.
Classes subclass of a value constraint in an object property	ISA07-ISA08	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF
Classes subclass of a cardinality constraint in an object property	ISA09-ISA12	DIFF	DIFF	N.E.	DIFF	DIFF	N.E.	N.E.	N.E.
Classes subclass of a cardinality constraint in a datatype property	ISA13-ISA16	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Classes subclass of a class intersection	ISA17	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	N.E.
<b>Class equivalences</b>									
Equivalent named classes	ISB01	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF
Classes equivalent to a value constraint in an object property	ISB02-ISB03	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF
Classes equivalent to a cardinality constraint in an object property	ISB04-ISB07	DIFF	DIFF	N.E.	DIFF	DIFF	N.E.	N.E.	N.E.
Classes equivalent to a cardinality constraint in a datatype property	ISB08-ISB11	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Classes equivalent to a class intersection	ISB12	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF
<b>Classes defined with set operators</b>									
Classes intersection of other classes	ISC01-ISC02	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF
<b>Property hierarchies</b>									
Object property hierarchies	ISD01-ISD02	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF
Datatype property hierarchies	ISD03-ISD04	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF
<b>Properties with domain and range</b>									
Object properties without domain or range	ISE01-ISE02	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	N.E.
Object properties with domain and range	ISE03-ISE04	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF
Object properties with multiple domains or ranges	ISE05-ISE06	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	N.E.
Datatype properties without domain or range	ISE07-ISE08	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Datatype properties with domain and range	ISE09	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF
Datatype properties with multiple domains	ISE10	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	N.E.
<b>Relations between properties</b>									
Equivalent object and datatype properties	ISF01-ISF02	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF
Inverse object properties	ISF03	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF
<b>Global cardinality constraints &amp; logical property characteristics</b>									
Transitive object properties	ISG01	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF
Symmetric object properties	ISG02	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF
Functional object and datatype properties	ISG03-ISG04	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF
Inverse functional object properties	ISG05	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	N.E.
<b>Individuals</b>									
Instances	ISH01, ISH03	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	N.E.
Instances of multiple classes	ISH02	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF
Named individuals and object properties	ISI01-ISI03	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF
Named individuals and datatype properties	ISI04-ISI05	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Anonymous individuals and object properties	ISJ01-ISJ02	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF
Anonymous individuals and datatype properties	ISJ03	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
<b>Individual identity</b>									
Equivalent individuals	ISK01	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF
Different individuals	ISK02-ISK03	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	N.E.	DIFF

Table 5.5: Summary of the results of the OWL interoperability of Protégé-Frames



Categories	Benchmarks	GA-PO	JE-PO	K2-PO	PF-PO	PO-PO	SP-PO	ST-PO	WE-PO
<b>Class hierarchies</b>									
Named class hierarchies without cycles	ISA01-ISA04	N.E.	SAME	DIFF	DIFF	SAME	SAME	DIFF	N.E.
Named class hierarchies with cycles	ISA05-ISA06	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	N.E.
Classes subclass of a value constraint in an object property	ISA07-ISA08	N.E.	SAME	SAME	DIFF	SAME	SAME	N.E.	DIFF
Classes subclass of a cardinality constraint in an object property	ISA09-ISA12	N.E.	SAME	N.E.	DIFF	SAME	SAME	N.E.	N.E.
Classes subclass of a cardinality constraint in a datatype property	ISA13-ISA16	N.E.	SAME	N.E.	N.E.	SAME	SAME	N.E.	N.E.
Classes subclass of a class intersection	ISA17	N.E.	SAME	SAME	DIFF	SAME	SAME	N.E.	N.E.
<b>Class equivalences</b>									
Equivalent named classes	ISB01	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Classes equivalent to a value constraint in an object property	ISB02-ISB03	N.E.	SAME	SAME	DIFF	SAME	SAME	N.E.	DIFF
Classes equivalent to a cardinality constraint in an object property	ISB04-ISB07	N.E.	SAME	N.E.	DIFF	SAME	SAME	N.E.	N.E.
Classes equivalent to a cardinality constraint in a datatype property	ISB08-ISB11	SAME	SAME	N.E.	N.E.	SAME	SAME	N.E.	N.E.
Classes equivalent to a class intersection	ISB12	SAME	SAME	SAME	DIFF	SAME	SAME	N.E.	DIFF
<b>Classes defined with set operators</b>									
Classes intersection of other classes	ISC01-ISC02	SAME	SAME	SAME	DIFF	SAME	SAME	N.E.	DIFF
<b>Property hierarchies</b>									
Object property hierarchies	ISD01-ISD02	DIFF	SAME	DIFF	DIFF	SAME	SAME	DIFF	DIFF
Datatype property hierarchies	ISD03-ISD04	N.E.	SAME	DIFF	DIFF	SAME	SAME	DIFF	DIFF
<b>Properties with domain and range</b>									
Object properties without domain or range	ISE01-ISE02	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	N.E.
Object properties with domain and range	ISE03-ISE04	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Object properties with multiple domains or ranges	ISE05-ISE06	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	N.E.
Datatype properties without domain or range	ISE07-ISE08	N.E.	SAME	SAME	N.E.	SAME	SAME	DIFF	DIFF
Datatype properties with domain and range	ISE09	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Datatype properties with multiple domains	ISE10	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	N.E.
<b>Relations between properties</b>									
Equivalent object and datatype properties	ISF01-ISF02	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Inverse object properties	ISF03	N.E.	DIFF	DIFF	DIFF	DIFF	DIFF	DIFF	DIFF
<b>Global cardinality constraints &amp; logical property characteristics</b>									
Transitive object properties	ISG01	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Symmetric object properties	ISG02	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Functional object and datatype properties	ISG03-ISG04	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Inverse functional object properties	ISG05	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	N.E.
<b>Individuals</b>									
Instances	ISH01, ISH03	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	N.E.
Instances of multiple classes	ISH02	DIFF	SAME	SAME	DIFF	SAME	SAME	SAME	DIFF
Named individuals and object properties	ISI01-ISI03	N.E.	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Named individuals and datatype properties	ISI04-ISI05	N.E.	SAME	SAME	N.E.	SAME	SAME	DIFF	DIFF
Anonymous individuals and object properties	ISJ01-ISJ02	N.E.	DIFF	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF
Anonymous individuals and datatype properties	ISJ03	N.E.	DIFF	DIFF	N.E.	DIFF	DIFF	N.E.	DIFF
<b>Individual identity</b>									
Equivalent individuals	ISK01	SAME	SAME	SAME	DIFF	SAME	SAME	DIFF	DIFF
Different individuals	ISK02-ISK03	SAME	SAME	SAME	DIFF	SAME	SAME	N.E.	DIFF

Table 5.6: Summary of the results of the OWL interoperability of Protégé-OWL

Categories	Benchmarks	GA-ST	JE-ST	K2-ST	PF-ST	PO-ST	SP-ST	ST-ST	WE-ST
<b>Class hierarchies</b>									
Named class hierarchies without cycles	ISA01-ISA04	N.E.	DIFF	DIFF	N.E.	DIFF	SAME	SAME	N.E.
Named class hierarchies with cycles	ISA05-ISA06	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF	DIFF	N.E.
Classes subclass of a value constraint in an object property	ISA07-ISA08	N.E.	N.E.	N.E.	N.E.	N.E.	DIFF	DIFF	DIFF
Classes subclass of a cardinality constraint in an object property	ISA09-ISA12	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Classes subclass of a cardinality constraint in a datatype property	ISA13-ISA16	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	DIFF	N.E.
Classes subclass of a class intersection	ISA17	N.E.	N.E.	N.E.	N.E.	N.E.	SAME	SAME	N.E.
<b>Class equivalences</b>									
Equivalent named classes	ISB01	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF	DIFF	DIFF
Classes equivalent to a value constraint in an object property	ISB02-ISB03	N.E.	N.E.	N.E.	N.E.	N.E.	DIFF	DIFF	DIFF
Classes equivalent to a cardinality constraint in an object property	ISB04-ISB07	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Classes equivalent to a cardinality constraint in a datatype property	ISB08-ISB11	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	DIFF	N.E.
Classes equivalent to a class intersection	ISB12	N.E.	N.E.	DIFF	N.E.	N.E.	DIFF	DIFF	DIFF
<b>Classes defined with set operators</b>									
Classes intersection of other classes	ISC01-ISC02	N.E.	N.E.	N.E.	N.E.	N.E.	SAME	SAME	DIFF
<b>Property hierarchies</b>									
Object property hierarchies	ISD01-ISD02	N.E.	DIFF	DIFF	N.E.	DIFF	SAME	SAME	DIFF
Datatype property hierarchies	ISD03-ISD04	DIFF	DIFF	DIFF	DIFF	DIFF	DIFF	DIFF	DIFF
<b>Properties with domain and range</b>									
Object properties without domain or range	ISE01-ISE02	N.E.	DIFF	SAME	N.E.	DIFF	SAME	SAME	N.E.
Object properties with domain and range	ISE03-ISE04	N.E.	DIFF	SAME	N.E.	DIFF	SAME	SAME	DIFF
Object properties with multiple domains or ranges	ISE05-ISE06	N.E.	DIFF	SAME	N.E.	DIFF	SAME	SAME	N.E.
Datatype properties without domain or range	ISE07-ISE08	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF	DIFF	DIFF
Datatype properties with domain and range	ISE09	N.E.	DIFF	DIFF	N.E.	DIFF	DIFF	DIFF	DIFF
Datatype properties with multiple domains	ISE10	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF	DIFF	N.E.
<b>Relations between properties</b>									
Equivalent object and datatype properties	ISF01-ISF02	N.E.	DIFF	DIFF	N.E.	DIFF	DIFF	DIFF	DIFF
Inverse object properties	ISF03	N.E.	DIFF	DIFF	N.E.	DIFF	DIFF	DIFF	DIFF
<b>Global cardinality constraints &amp; logical property characteristics</b>									
Transitive object properties	ISG01	DIFF	DIFF	SAME	N.E.	DIFF	SAME	SAME	DIFF
Symmetric object properties	ISG02	N.E.	DIFF	SAME	N.E.	DIFF	SAME	SAME	DIFF
Functional object and datatype properties	ISG03-ISG04	N.E.	N.E.	DIFF	N.E.	DIFF	DIFF	DIFF	DIFF
Inverse functional object properties	ISG05	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF	DIFF	N.E.
<b>Individuals</b>									
Instances	ISH01, ISH03	N.E.	SAME	SAME	N.E.	SAME	SAME	SAME	N.E.
Instances of multiple classes	ISH02	N.E.	DIFF	SAME	N.E.	DIFF	SAME	SAME	DIFF
Named individuals and object properties	ISI01-ISI03	N.E.	DIFF	SAME	N.E.	DIFF	SAME	SAME	DIFF
Named individuals and datatype properties	ISI04-ISI05	N.E.	DIFF	DIFF	N.E.	DIFF	DIFF	DIFF	DIFF
Anonymous individuals and object properties	ISJ01-ISJ02	N.E.	N.E.	DIFF	N.E.	N.E.	DIFF	DIFF	DIFF
Anonymous individuals and datatype properties	ISJ03	N.E.	N.E.	DIFF	N.E.	N.E.	DIFF	DIFF	DIFF
<b>Individual identity</b>									
Equivalent individuals	ISK01	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF	DIFF	DIFF
Different individuals	ISK02-ISK03	N.E.	N.E.	N.E.	N.E.	N.E.	DIFF	DIFF	N.E.

Table 5.7: Summary of the results of the OWL interoperability of SemTalk

Categories	Benchmarks	GA-SP	JE-SP	K2-SP	PF-SP	PO-SP	SP-SP	ST-SP	WE-SP
<b>Class hierarchies</b>									
Named class hierarchies without cycles	ISA01-ISA04	N.E.	SAME	DIFF	N.E.	SAME	SAME	SAME	N.E.
Named class hierarchies with cycles	ISA05-ISA06	N.E.	SAME	SAME	N.E.	SAME	SAME	DIFF	N.E.
Classes subclass of a value constraint in an object property	ISA07-ISA08	N.E.	SAME	SAME	N.E.	SAME	SAME	DIFF	N.E.
Classes subclass of a cardinality constraint in an object property	ISA09-ISA12	N.E.	SAME	N.E.	N.E.	SAME	SAME	N.E.	N.E.
Classes subclass of a cardinality constraint in a datatype property	ISA13-ISA16	N.E.	SAME	N.E.	N.E.	SAME	SAME	N.E.	N.E.
Classes subclass of a class intersection	ISA17	SAME	SAME	SAME	N.E.	SAME	SAME	SAME	N.E.
<b>Class equivalences</b>									
Equivalent named classes	ISB01	SAME	SAME	SAME	N.E.	SAME	SAME	DIFF	N.E.
Classes equivalent to a value constraint in an object property	ISB02-ISB03	N.E.	SAME	SAME	N.E.	SAME	SAME	DIFF	N.E.
Classes equivalent to a cardinality constraint in an object property	ISB04-ISB07	N.E.	SAME	N.E.	N.E.	SAME	SAME	N.E.	N.E.
Classes equivalent to a cardinality constraint in a datatype property	ISB08-ISB11	N.E.	SAME	N.E.	N.E.	SAME	SAME	N.E.	N.E.
Classes equivalent to a class intersection	ISB12	N.E.	SAME	SAME	N.E.	SAME	SAME	DIFF	N.E.
<b>Classes defined with set operators</b>									
Classes intersection of other classes	ISC01-ISC02	N.E.	SAME	SAME	N.E.	SAME	SAME	SAME	N.E.
<b>Property hierarchies</b>									
Object property hierarchies	ISD01-ISD02	N.E.	SAME	DIFF	N.E.	SAME	SAME	SAME	N.E.
Datatype property hierarchies	ISD03-ISD04	N.E.	SAME	DIFF	N.E.	SAME	SAME	DIFF	DIFF
<b>Properties with domain and range</b>									
Object properties without domain or range	ISE01-ISE02	N.E.	SAME	SAME	N.E.	SAME	SAME	SAME	N.E.
Object properties with domain and range	ISE03-ISE04	N.E.	SAME	SAME	N.E.	SAME	SAME	SAME	N.E.
Object properties with multiple domains or ranges	ISE05-ISE06	SAME	SAME	SAME	N.E.	SAME	SAME	SAME	N.E.
Datatype properties without domain or range	ISE07-ISE08	N.E.	SAME	SAME	N.E.	SAME	SAME	DIFF	N.E.
Datatype properties with domain and range	ISE09	SAME	SAME	SAME	N.E.	SAME	SAME	DIFF	N.E.
Datatype properties with multiple domains	ISE10	SAME	SAME	SAME	N.E.	SAME	SAME	DIFF	N.E.
<b>Relations between properties</b>									
Equivalent object and datatype properties	ISF01-ISF02	SAME	SAME	SAME	N.E.	SAME	SAME	DIFF	N.E.
Inverse object properties	ISF03	SAME	SAME	SAME	N.E.	DIFF	SAME	DIFF	N.E.
<b>Global cardinality constraints &amp; logical property characteristics</b>									
Transitive object properties	ISG01	N.E.	SAME	SAME	N.E.	SAME	SAME	SAME	N.E.
Symmetric object properties	ISG02	SAME	SAME	SAME	N.E.	SAME	SAME	SAME	N.E.
Functional object and datatype properties	ISG03-ISG04	N.E.	SAME	SAME	N.E.	SAME	SAME	DIFF	N.E.
Inverse functional object properties	ISG05	SAME	SAME	SAME	N.E.	SAME	SAME	DIFF	N.E.
<b>Individuals</b>									
Instances	ISH01, ISH03	N.E.	SAME	SAME	N.E.	SAME	SAME	SAME	N.E.
Instances of multiple classes	ISH02	N.E.	SAME	SAME	N.E.	SAME	SAME	SAME	N.E.
Named individuals and object properties	ISI01-ISI03	N.E.	SAME	SAME	N.E.	SAME	SAME	SAME	N.E.
Named individuals and datatype properties	ISI04-ISI05	N.E.	SAME	SAME	N.E.	SAME	SAME	DIFF	N.E.
Anonymous individuals and object properties	ISJ01-ISJ02	N.E.	DIFF	DIFF	N.E.	DIFF	DIFF	DIFF	N.E.
Anonymous individuals and datatype properties	ISJ03	N.E.	DIFF	DIFF	N.E.	DIFF	DIFF	DIFF	N.E.
<b>Individual identity</b>									
Equivalent individuals	ISK01	SAME	SAME	SAME	N.E.	SAME	SAME	DIFF	N.E.
Different individuals	ISK02-ISK03	N.E.	SAME	SAME	N.E.	SAME	SAME	DIFF	N.E.

Table 5.8: Summary of the results of the OWL interoperability of SWI-Prolog

Categories	Benchmarks	GA-WE	JE-WE	K2-WE	PF-WE	PO-WE	SP-WE	ST-WE	WE-WE
<b>Class hierarchies</b>									
Named class hierarchies without cycles	ISA01-ISA04	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Named class hierarchies with cycles	ISA05-ISA06	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Classes subclass of a value constraint in an object property	ISA07-ISA08	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF
Classes subclass of a cardinality constraint in an object property	ISA09-ISA12	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Classes subclass of a cardinality constraint in a datatype property	ISA13-ISA16	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Classes subclass of a class intersection	ISA17	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
<b>Class equivalences</b>									
Equivalent named classes	ISB01	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF
Classes equivalent to a value constraint in an object property	ISB02-ISB03	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF
Classes equivalent to a cardinality constraint in an object property	ISB04-ISB07	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Classes equivalent to a cardinality constraint in a datatype property	ISB08-ISB11	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Classes equivalent to a class intersection	ISB12	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF
<b>Classes defined with set operators</b>									
Classes intersection of other classes	ISC01-ISC02	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF
<b>Property hierarchies</b>									
Object property hierarchies	ISD01-ISD02	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF
Datatype property hierarchies	ISD03-ISD04	N.E.	DIFF	DIFF	DIFF	DIFF	DIFF	DIFF	DIFF
<b>Properties with domain and range</b>									
Object properties without domain or range	ISE01-ISE02	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Object properties with domain and range	ISE03-ISE04	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF
Object properties with multiple domains or ranges	ISE05-ISE06	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Datatype properties without domain or range	ISE07-ISE08	N.E.	DIFF	DIFF	N.E.	DIFF	N.E.	DIFF	N.E.
Datatype properties with domain and range	ISE09	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF	N.E.
Datatype properties with multiple domains	ISE10	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
<b>Relations between properties</b>									
Equivalent object and datatype properties	ISF01-ISF02	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF	N.E.
Inverse object properties	ISF03	N.E.	DIFF	DIFF	DIFF	DIFF	DIFF	DIFF	DIFF
<b>Global cardinality constraints &amp; logical property characteristics</b>									
Transitive object properties	ISG01	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF
Symmetric object properties	ISG02	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF
Functional object and datatype properties	ISG03-ISG04	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF	N.E.
Inverse functional object properties	ISG05	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
<b>Individuals</b>									
Instances	ISH01, ISH03	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.	N.E.
Instances of multiple classes	ISH02	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF
Named individuals and object properties	ISI01-ISI03	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF
Named individuals and datatype properties	ISI04-ISI05	N.E.	DIFF	DIFF	N.E.	DIFF	N.E.	DIFF	N.E.
Anonymous individuals and object properties	ISJ01-ISJ02	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF
Anonymous individuals and datatype properties	ISJ03	N.E.	DIFF	DIFF	N.E.	DIFF	N.E.	DIFF	N.E.
<b>Individual identity</b>									
Equivalent individuals	ISK01	N.E.	DIFF	DIFF	DIFF	DIFF	N.E.	DIFF	DIFF
Different individuals	ISK02-ISK03	N.E.	DIFF	N.E.	DIFF	DIFF	N.E.	N.E.	DIFF

Table 5.9: Summary of the results of the OWL interoperability of WebODE

### 5.2.2 Interoperability issues

The results of the interoperability of the tools participating in the benchmarking depend not just on the behaviour of the tools during the import and export operation (as described in each section that deals with the tools) but also on the following issues identified in the results:

- In the case of interchanges **from KAON2 to GATE**, when GATE uses ontologies generated by KAON2, it produces ontologies that make the comparer execution fail. This is sometime because the ontologies are not valid OWL ontologies in the RDF/XML syntax.
- In the case of interchanges **from GATE to Jena, KAON2 and Protégé-OWL**, when Jena, KAON2 and Protégé-OWL use ontologies generated by GATE, they produce ontologies that make the comparer execution fail.
- In the case of interchanges **from Protégé-Frames to GATE**, when GATE uses ontologies generated by Protégé-Frames, when the ontologies include classes with multiple instances, GATE produces ontologies that make the comparer execution fail.
- In the case of interchanges **from Protégé-Frames to SemTalk**, when SemTalk uses ontologies generated by Protégé-Frames, it produces ontologies that make the comparer execution fail.
- In the case of interchanges **from SemTalk and SWI-Prolog to GATE**, when GATE uses ontologies generated by SemTalk and SWI-Prolog, it produces ontologies that make the comparer execution fail.
- In the case of interchanges **from SemTalk to Jena**, when Jena uses ontologies generated by SemTalk, it loses the datatype property hierarchies.
- In the case of interchanges **from SemTalk to KAON2**, when KAON2 uses ontologies generated by SemTalk, it produces ontologies that make the comparer execution fail.
- In the case of interchanges **from GATE, Jena and Protégé-OWL to SemTalk**, when SemTalk uses ontologies generated by these tools, it: loses the *rdfs:subClassOf* and *rdfs:subPropertyOf* properties; loses the domain and the range in object or datatype properties with domain or range; its execution fails with classes that are a subclass or equivalent to value constraints, cardinality constraints, or class intersections; its execution fails with classes intersection of other classes; and its execution fails with anonymous individuals with object or datatype properties.

- In the case of interchanges **from Protégé-Frames and WebODE to SWI-Prolog**, SWI-Prolog generates OWL ontologies that are not valid in the RDF/XML syntax from most of the ontologies produced by WebODE and from all the ontologies produced by Protégé-Frames.

### 5.2.3 Components that can be interchanged between the tools

Taking into account these issues, we present a summary of the combinations of components that can only be interchanged between the tools participating in the benchmarking.

#### **Jena - Protégé-OWL - SWI-Prolog**

These tools can interchange any combination of components.

#### **KAON2 - KAON2, KAON2 - Jena, KAON2 - Protégé-OWL, and KAON2 - SWI-Prolog**

- Named class hierarchies with cycles.
- Classes that are a subclass of a value constraint in an object property.
- Classes that are a subclass of a class intersection.
- Equivalent named classes.
- Classes equivalent to a value constraint in an object property.
- Classes equivalent to a class intersection.
- Classes intersection of other classes.
- Object and datatype properties with or without domain or range, or with multiple domains or ranges.
- Equivalent object and datatype properties.
- Inverse, transitive, symmetric and inverse functional object properties.
- Functional datatype properties.
- Instances of single and multiple classes.
- Named and anonymous individuals and object or datatype properties.
- Equivalent and different individuals.

### **SemTalk - SemTalk and SemTalk - SWI-Prolog**

- Named class hierarchies without cycles.
- Classes that are a subclass of a class intersection.
- Classes intersection of other classes.
- Object property hierarchies.
- Object properties with or without domain or range or with multiple domains and ranges.
- Transitive and symmetric object properties.
- Instances of single and multiple classes.
- Named individuals and object properties.

### **GATE - SWI-Prolog**

- Classes that are a subclass of a class intersection.
- Equivalent named classes.
- Object properties with multiple domains or ranges.
- Datatype properties with domain and range.
- Datatype properties with multiple domains.
- Symmetric and inverse functional object properties.
- Equivalent individuals.

### **GATE - JENA**

- Equivalent named classes.
- Classes equivalent to a cardinality constraint in a datatype property.
- Classes equivalent to a class intersection.
- Classes intersection of other classes.
- Inverse and inverse functional object properties.
- Equivalent and different individuals.

**GATE - Protégé-OWL**

- Equivalent named classes.
- Classes equivalent to a cardinality constraint in a datatype property.
- Classes equivalent to a class intersection.
- Classes intersection of other classes.
- Symmetric and inverse functional object properties.
- Equivalent and different individuals.

**GATE - GATE**

- Equivalent named classes.
- Classes equivalent to a class intersection.
- Datatype properties with multiple domains.
- Transitive and inverse functional object properties.

**KAON2 - SemTalk**

- Object properties with or without domain or range or with multiple domains and ranges.
- Transitive and symmetric object properties.
- Instances of single and multiple classes.
- Named individuals and object properties.

**GATE - KAON2**

- Classes that are a subclass or equivalent to a class intersection.
- Datatype properties with domain and range or with multiple domains.
- Transitive and inverse functional object properties.

**SemTalk - Jena and SemTalk - Protégé-OWL**

- Instances of multiple classes.



### **SemTalk - GATE**

These tools cannot interchange any combination of components.

### **Protégé-Frames - all**

These tools cannot interchange any combination of components.

### **WebODE - all**

Taking into account these issues, WebODE cannot interchange with the other tools any combination of components.

# Chapter 6

## Conclusion

*by* RAÚL GARCÍA-CASTRO

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

The main goal fulfilled with this work is the assessment of the current interoperability of nine best-in-class Semantic Web tools. The assessment has provided us with results about the detailed behaviour of the tools not just when interoperating with other tools but also when importing and exporting OWL ontologies. We have also developed the IBSE tool, an easy-to-use tool for large-scale evaluations of the interoperability of the Semantic Web technology when using an interchange language.

As in the case of the RDF(S) Interoperability Benchmarking, the benchmarking process has been long. And as a result, we have discovered that the interoperability between the tools is very low and that real interoperability in the Semantic Web requires the involvement of tool developers. In some cases, this is due to the representation formalisms managed by the tools, but in other cases it is due to defects in the tools or to the way of serializing the ontologies, which has a high impact in interoperability.

This panoramic, although disappointing, can serve to promote the second of our goals: the improvement of the tools. Although this goal is out of our scope just now because each tool is developed by independent organizations, we hope, nevertheless, that the results we provide may help in their improvement.

The benchmarking results are now publicly available in the Web in machine-processable format. Thus, anyone can use them for comparing them with their own results or for reasoning about them.

The tool developers that have participated in this benchmarking will receive the final version of this document, although they are already informed about the recommendations proposed for improving their tools.

Any developer of Semantic Web tools can benefit from this work by learning the cor-

rect or incorrect behaviour of the other tools. They can also use the IBSE tool to evaluate their tools, either in the early stages of their development or when the development has finished, and to monitor their improvement.

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

The IBSE tool can also be used in other scenarios. It can be used for evaluating the interoperability of tools using other languages as interchange. Right now the tool allows performing experiments using RDF(S) as interchange language. Other tools should have to implement the corresponding method in the IBSE tool and then use the RDF(S) Import Benchmark Suite<sup>1</sup> as ontology dataset. The IBSE tool can also be used to evaluate the importers and exporters of any tool because the interoperability results (even of one tool with itself) provide useful insights about the behaviour of the tool importers and exporters.

---

<sup>1</sup>[http://knowledgeweb.semanticweb.org/benchmarking\\_interoperability/rdfs/rdfs\\_import\\_benchmark\\_suite.html](http://knowledgeweb.semanticweb.org/benchmarking_interoperability/rdfs/rdfs_import_benchmark_suite.html)

# Appendix A

## List of benchmarks of the OWL Lite Import Benchmark Suite

This appendix contains a list of the benchmarks that compose the OWL Import Benchmark Suite, which are described by:

- A unique identifier (i.e., **ISA01** where **IS** denotes the OWL import benchmark suite, **A** is the group to which the benchmark belongs to, and **01** is a number)
- A description of the ontology in natural language (e.g., *Import a single class*).
- The description of the ontology in the Description Logics formalism. All these descriptions can be found in Appendix B.
- A graphical representation of the ontology, that uses the notation shown in Figure A.1

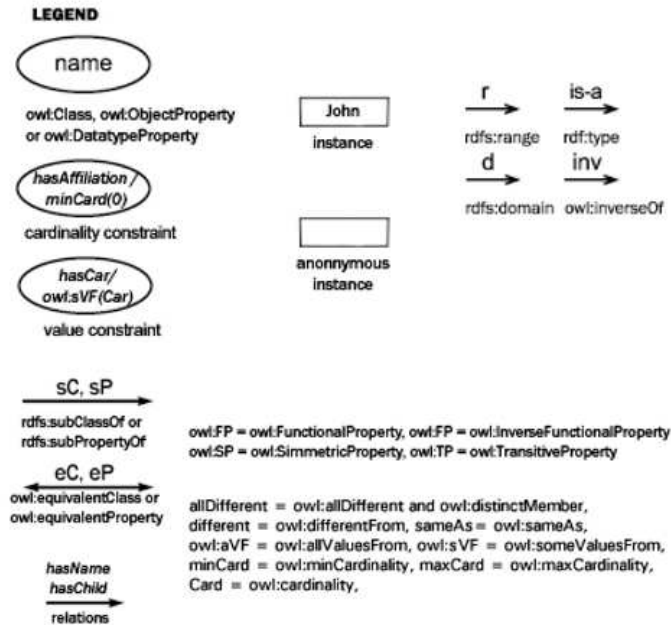


Figure A.1: Notation used in the benchmarks

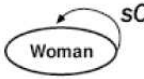
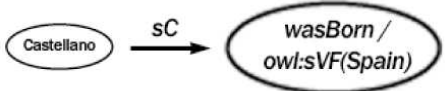
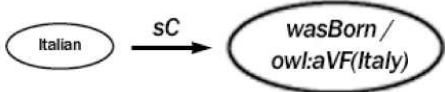
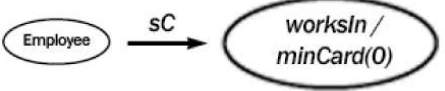
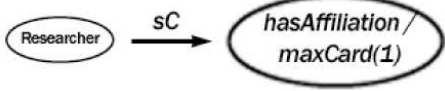
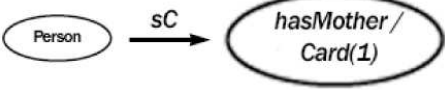
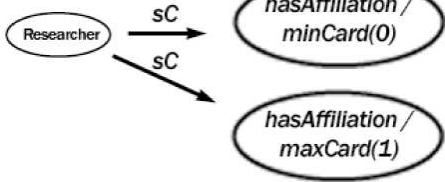
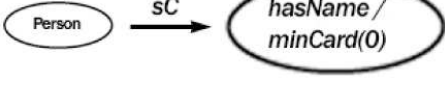
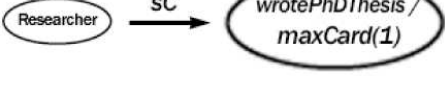
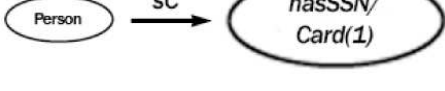
## Class benchmarks

### Group A: Class hierarchies

ID	Description	Graphical representation
ISA01	Import a single class	Person
ISA02	Import a single class, subclass of a second class which is subclass of a third one	Child $\xrightarrow{sC}$ Man $\xrightarrow{sC}$ Person
ISA03	Import a class that is subclass of two classes	Child $\xrightarrow{sC}$ Man Child $\xrightarrow{sC}$ Person
ISA04	Import several classes subclass of a single class	Man $\xrightarrow{sC}$ Person Woman $\xrightarrow{sC}$ Person
ISA05	Import two classes, each subclass of the other	Man $\xrightarrow{sC}$ Male Male $\xrightarrow{sC}$ Man

(continued on next page)

*(continued from previous page)*

ISA06	Import a class, subclass of itself	
ISA07	Import a class which is subclass of an anonymous class defined by an owl:someValuesFrom value constraint in an object property	
ISA08	Import a class which is subclass of an anonymous class defined by an owl:allValuesFrom value constraint in an object property	
ISA09	Import a class which is subclass of an anonymous class defined by an owl:minCardinality=0 cardinality constraint in an object property	
ISA10	Import a class which is subclass of an anonymous class defined by an owl:maxCardinality=1 cardinality constraint in an object property	
ISA11	Import a class which is subclass of an anonymous class defined by an owl:cardinality=1 cardinality constraint in an object property	
ISA12	Import a class which is subclass of an anonymous class defined by an owl:minCardinality=0 and an owl:maxCardinality=1 cardinality constraints in an object property	
ISA13	Import a class which is subclass of an anonymous class defined by an owl:minCardinality=0 cardinality constraint in a datatype property	
ISA14	Import a class which is subclass of an anonymous class defined by an owl:maxCardinality=1 cardinality constraint in a datatype property	
ISA15	Import a class which is subclass of an anonymous class defined by an owl:cardinality=1 cardinality constraint in a datatype property	

*(continued on next page)*

(continued from previous page)

ISA16	Import a class which is subclass of an anonymous class defined by an owl:minCardinality=0 and an owl:maxCardinality=1 cardinality constraints in a datatype property	
ISA17	Import a class which is subclass of an anonymous class defined by the intersection of two other classes	

**Group B: Class equivalences**

ID	Description	Graphical representation
ISB01	Import several classes which are all of them equivalent	
ISB02	Import a class which is equivalent to an anonymous class defined by an owl:someValuesFrom value constraint in an object property	
ISB03	Import a class which is equivalent to an anonymous class defined by an owl:allValuesFrom value constraint in an object property	
ISB04	Import a class which is equivalent to an anonymous class defined by an owl:minCardinality=0 cardinality constraint in an object property	
ISB05	Import a class which is equivalent to an anonymous class defined by an owl:maxCardinality=1 cardinality constraint in an object property	
ISB06	Import a class which is equivalent to an anonymous class defined by an owl:cardinality=1 cardinality constraint in an object property	

(continued on next page)

(continued from previous page)

ISB07	Import a class which is equivalent to an anonymous class defined by an owl:minCardinality=0 and an owl:maxCardinality=1 cardinality constraints in an object property	
ISB08	Import a class which is equivalent to an anonymous class defined by an owl:minCardinality=0 cardinality constraint in a datatype property	
ISB09	Import a class which is equivalent to an anonymous class defined by an owl:maxCardinality=1 cardinality constraint in a datatype property	
ISB10	Import a class which is equivalent to an anonymous class defined by an owl:cardinality=1 cardinality constraint in a datatype property	
ISB11	Import a class which is equivalent to an anonymous class defined by an owl:minCardinality=0 and an owl:maxCardinality=1 cardinality constraints in a datatype property	
ISB12	Import a class which is equivalent to an anonymous class defined by the intersection of two other classes	


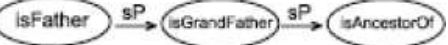

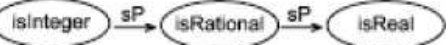
### Group C: Classes defined with set operators

ID	Description	Graphical representation
ISC01	Import a class which is intersection of two other classes	
ISC02	Import a class which is intersection of several other classes	

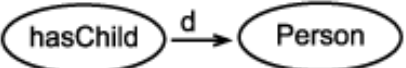
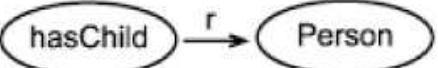
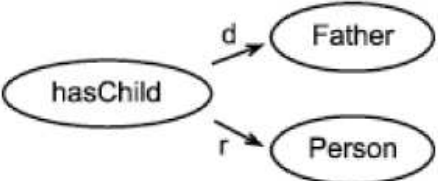
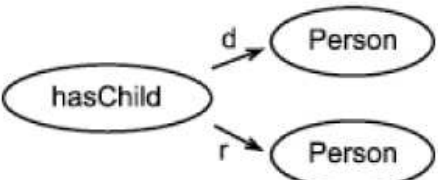


## Property benchmarks

### Group D: Property hierarchies

ID	Description	Graphical representation
ISD01	Import a single object property	
ISD02	Import an object property that is subproperty of another object property that is subproperty of a third one	
ISD03	Import a single datatype property	
ISD04	Import a datatype property that is subproperty of another datatype property that is subproperty of a third one	

### Group E: Properties with domain and range

ID	Description	Graphical representation
ISE01	Import a single object property with domain a class	
ISE02	Import a single object property with range a class	
ISE03	Import a single object property with domain a class and range another class	
ISE04	Import a single object property with domain and range the same class	

(continued on next page)

*(continued from previous page)*

ISE05	Import a single object property with domain multiple classes and range a class	
ISE06	Import a single object property with domain a class and range multiple classes	
ISE07	Import a single datatype property with domain a class	
ISE08	Import a single datatype property with range <code>rdfs:Literal</code>	
ISE09	Import a single datatype property with domain a class and range <code>rdfs:Literal</code>	
ISE10	Import a single datatype property with domain multiple classes and range <code>rdfs:Literal</code>	

## Group F: Relations between properties

ID	Description	Graphical representation
----	-------------	--------------------------

*(continued on next page)*

(continued from previous page)

ISF01	Import several object properties with domain a class and range another class, which are all of them equivalent	
ISF02	Import several datatype properties with domain a class and range <code>rdfs:Literal</code> , which are all of them equivalent	
ISF03	Import an object property with domain a class and range another class, which is inverse of another object property	

### Group G: Global cardinality constraints and logical property characteristics

ID	Description	Graphical representation
ISG01	Import a single transitive object property with domain and range the same class	
ISG02	Import a single symmetric object property with domain and range the same class	

(continued on next page)

*(continued from previous page)*

ISG03	Import a single functional object property with domain a class and range another class	
ISG04	Import a single functional datatype property with domain a class and range <code>rdfs:Literal</code>	
ISG05	Import a single inverse functional object property with domain a class and range another class	

## Individual benchmarks

### Group H: Single individuals

ID	Description	Graphical representation
ISH01	Import one class and one individual that is instance of the class	

*(continued on next page)*

(continued from previous page)

ISH02	Import several classes and one individual that is instance of all of them	<pre> graph LR     Peter[Peter] -- is-a --&gt; Student((Student))     Peter -- is-a --&gt; Father((Father))     Peter -- is-a --&gt; Person((Person)) </pre>
ISH03	Import one class and several individuals that are instance of the class	<pre> graph LR     Peter[Peter] -- is-a --&gt; Person((Person))     Paul[Paul] -- is-a --&gt; Person     Mary[Mary] -- is-a --&gt; Person </pre>

### Group I: Named individuals and properties

ID	Description	Graphical representation
ISI01	Import one class, one object property with domain and range the class, and one individual of the class that has the object property with another individual of the same class	<pre> graph LR     Mary[Mary] -- is-a --&gt; Person1((Person))     Paul[Paul] -- is-a --&gt; Person2((Person))     Mary -- hasChild --&gt; Paul     subgraph Property         direction TB         d1[d]         r1[r]         hasChild1(hasChild)         d1 --&gt; hasChild1         r1 --&gt; hasChild1     end </pre>
ISI02	Import one class, one object property with domain and range the class, and one individual of the class that has the object property with himself	<pre> graph LR     Peter[Peter] -- is-a --&gt; Person((Person))     Peter -- knows --&gt; Peter     subgraph Property         direction TB         d2[d]         r2[r]         knows((knows))         d2 --&gt; knows         r2 --&gt; knows     end </pre>
ISI03	Import two classes, one object property with domain one class and range the other class, and one individual of one class that has the object property with an individual of the other class	<pre> graph LR     Mary[Mary] -- is-a --&gt; Mother((Mother))     Paul[Paul] -- is-a --&gt; Child((Child))     Mary -- hasChild --&gt; Paul     subgraph Property         direction TB         d3[d]         r3[r]         hasChild2(hasChild)         d3 --&gt; hasChild2         r3 --&gt; hasChild2     end </pre>

(continued on next page)

*(continued from previous page)*

ISI04	Import one class, one datatype property with domain the class and range <code>rdfs:Literal</code> , and one individual of the class that has the datatype property with a literal	
ISI05	Import one class, one datatype property with domain the class and range <code>rdfs:Literal</code> , and one individual of the class that has the datatype property with several literals	

## Group J: Anonymous individuals and properties

ID	Description	Graphical representation
ISJ01	Import one class, one object property with domain and range the class, and one anonymous individual of the class that has the object property with another individual of the same class	
ISJ02	Import two classes, one object property with domain one class and range the other class, and one anonymous individual of one class that has the object property with an individual of the other class	
ISJ03	Import one class, one datatype property with domain the class and range <code>rdfs:Literal</code> , and one anonymous individual of the class that has the datatype property with a literal	

**Group K: Individual identity**

ID	Description	Graphical representation
ISK01	Import one class and two named individuals of the class that are the same	
ISK02	Import one class and two named individuals of the class that are the different	
ISK03	Import one class and three named individuals of the class that are all of them different	

**Syntax and abbreviation benchmarks****Group L: Syntax and abbreviation benchmarks**

ID	Description
ISL01	Import several resources with absolute URI references
ISL02	Import several resources with URI references relative to a base URI
ISL03	Import several resources with URI references transformed from rdf:ID attribute values
ISL04	Import several resources with URI references relative to an ENTITY declaration
<b>Empty node benchmarks</b>	
ISL05	Import several resources with empty nodes
ISL06	Import several resources with empty nodes shortened
<b>Multiple properties benchmarks</b>	
ISL07	Import several resources with multiple properties
ISL08	Import several resources with multiple properties shortened
<b>Empty node benchmarks</b>	

ISL09	Import several resources with typed nodes
ISL10	Import several resources with typed nodes shortened
<b>Empty node benchmarks</b>	
ISL11	Import several resources with properties with string literals
ISL12	Import several resources with properties with string literals as XML attributes
<b>Empty node benchmarks</b>	
ISL13	Import several resources with blank nodes with identifier
ISL14	Import several resources with blank nodes shortened
<b>Language identification benchmarks</b>	
ISL15	Import several resources with properties with xml:lang attributes



# Appendix B

## Description of the ontologies in DL

This appendix provides a formal description of the ontologies that compose the OWL Lite Import Benchmark Suite<sup>1</sup>, in the Description Logics formalism.

The formalism presented in this appendix adopts the following conventional notation, presented in [Volz, 2004], to map the OWL axioms in the abstract syntax to Description Logics concepts.

Axiom	DL
Class ( <i>C</i> partial $D_1 \dots D_n$ ) Class ( <i>C</i> complete $D_1 \dots D_n$ ) DisjointClasses( $C_1 \dots C_n$ ) EquivalentClasses( $C_1 \dots C_n$ ) SubClassOf( $C_1 C_2$ )	$C \sqsubseteq (D_1 \sqcap \dots \sqcap D_n)$ $C \equiv (D_1 \sqcap \dots \sqcap D_n)$ $C_1 \sqsubseteq \neg C_n$ $(C_1 \equiv C_n)$ $(C_1 \sqsubseteq C_2)$
Property( <i>P</i> domain( $D_1 \dots D_n$ ) range( $D_1 \dots D_n$ ) super( $Q_1 \dots Q_n$ ) inverseOf <i>Q</i> Symmetric Transitive Functional InverseFunctional )	$\top \sqsubseteq \forall P^-.D_i; \forall 1 \leq i \leq n$ $\top \sqsubseteq \forall P.D_i; \forall 1 \leq i \leq n$ $P \sqsubseteq Q_i; \forall 1 \leq i \leq n$ $P \equiv Q^-$ $P \equiv P^-$ $P^+ \sqsubseteq P$ $\top \sqsubseteq \forall P$ $\top \sqsubseteq \forall P^-$
SameIndividuals( $(o_1 \dots o_n)$ ) DifferentIndividuals( $(D_1 \dots D_n)$ )	$(o_1 = o_i); \forall 1 \leq i \leq n$ $\neg(o_i = o_j); \forall 1 \leq i \leq j \leq n$

<sup>1</sup>[http://knowledgeweb.semanticweb.org/benchmarking\\_interoperability/owl/import.html](http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/import.html)

On the left side it appears the abstract syntax of an OWL axiom and on the right side the corresponding axiom expressed in the Description Logics formalism.

Table B.1 shows a sample description of an ontology defined in the OWL Lite Import Benchmark Suite: each entry comes with a description in both natural language and in the Description Logics formalism.

ID	Here there is the description in natural language... ...and here the one in the Description Logics formalism.
ISE06	Import a single object property with domain a class and range multiple classes $\top \sqsubseteq \forall hasChild^-. Person$ $\top \sqsubseteq \forall hasChild. Person$ $\top \sqsubseteq \forall hasChild. Human$ $\top \sqsubseteq \forall hasChild. Child$

Table B.1: Structure of the tables and a sample instantiation

## Class benchmarks

### Group A: Class hierarchies

ISA01	Import a single class $Person$
ISA02	Import a single class, subclass of a second class which is subclass of a third one $Child \sqsubseteq Man \sqsubseteq Person$
ISA03	Import a class that is subclass of two classes $Child \sqsubseteq Man$ $Child \sqsubseteq Person$
ISA04	Import several classes subclass of a single class $Woman \sqsubseteq Person$ $Man \sqsubseteq Person$
ISA05	Import two classes, each subclass of the other $Male \sqsubseteq Man$ $Man \sqsubseteq Male$
ISA06	Import a class, subclass of itself $Woman \sqsubseteq Woman$

(continued on next page)

(continued from previous page)

<b>ISA07</b>	Import a class which is subclass of an anonymous class defined by an owl:someValuesFrom value constraint in an object property <div>Driver <math>\sqsubseteq \exists hasCar.Car</math></div>
<b>ISA08</b>	Import a class which is subclass of an anonymous class defined by an owl:allValuesFrom value constraint in an object property <div>Italian <math>\sqsubseteq \forall wasBorn.Italy</math></div>
<b>ISA09</b>	Import a class which is subclass of an anonymous class defined by an owl:minCardinality=0 cardinality constraint in an object property <div>Employee <math>\sqsubseteq \geq 0 worksIn</math></div>
<b>ISA10</b>	Import a class which is subclass of an anonymous class defined by an owl:maxCardinality=1 cardinality constraint in an object property <div>Researcher <math>\sqsubseteq \leq 1 hasAffiliation</math></div>
<b>ISA11</b>	Import a class which is subclass of an anonymous class defined by an owl:cardinality=1 cardinality constraint in an object property <div>Man <math>\sqsubseteq = 1 hasMother</math></div>
<b>ISA12</b>	Import a class which is subclass of an anonymous class defined by an owl:minCardinality=0 and an owl:maxCardinality=1 cardinality constraints in an object property <div>Researcher <math>\sqsubseteq \geq 0 hasAffiliation</math> Researcher <math>\sqsubseteq \leq 1 hasAffiliation</math></div>
<b>ISA13</b>	Import a class which is subclass of an anonymous class defined by an owl:minCardinality=0 cardinality constraint in a datatype property <div>Person <math>\sqsubseteq \geq 0 hasName</math></div>
<b>ISA14</b>	Import a class which is subclass of an anonymous class defined by an owl:maxCardinality=1 cardinality constraint in a datatype property <div>Researcher <math>\sqsubseteq \leq 1 wrotePhDThesis</math></div>
<b>ISA15</b>	Import a class which is subclass of an anonymous class defined by an owl:cardinality=1 cardinality constraint in a datatype property <div>Person <math>\sqsubseteq = 1 hasSSN</math></div>

(continued on next page)

(continued from previous page)

<b>ISA16</b>	Import a class which is subclass of an anonymous class defined by an <code>owl:minCardinality=0</code> and an <code>owl:maxCardinality=1</code> cardinality constraints in a datatype property
	$\text{Researcher} \sqsubseteq \geq 0 \text{ wrotePhDThesis}$ $\text{Researcher} \sqsubseteq \leq 1 \text{ wrotePhDThesis}$
<b>ISA17</b>	Import a class which is subclass of a class defined by the intersection of two other classes
	$\text{ItalianMan} \sqsubseteq (\text{Italian} \sqcap \text{Male})$

**Group B: Class Equivalences**

<b>ISB01</b>	Import several classes which are all of them equivalent
	$\text{Italian} \equiv \text{Italiano} \equiv \text{Italienne}$
<b>ISB02</b>	Import a class which is equivalent to an anonymous class defined by an <code>owl:someValuesFrom</code> value constraint in an object property
	$\text{Driver} \equiv \exists \text{ hasCar. Car}$
<b>ISB03</b>	Import a class which is equivalent to an anonymous class defined by an <code>owl:allValuesFrom</code> value constraint in an object property
	$\text{Italian} \equiv \forall \text{ wasBorn. Italy}$
<b>ISB04</b>	Import a class which is equivalent to an anonymous class defined by an <code>owl:minCardinality=1</code> cardinality constraint in an object property
	$\text{Employee} \equiv \geq 1 \text{ worksIn}$
<b>ISB05</b>	Import a class which is equivalent to an anonymous class defined by an <code>owl:maxCardinality=1</code> cardinality constraint in an object property
	$\text{Researcher} \equiv \leq 1 \text{ hasAffiliation}$
<b>ISB06</b>	Import a class which is equivalent to an anonymous class defined by an <code>owl:cardinality=1</code> cardinality constraint in an object property
	$\text{Man} \equiv = 1 \text{ hasMother}$

(continued on next page)

(continued from previous page)

<b>ISB07</b>	Import a class which is equivalent to an anonymous class defined by an <code>owl:minCardinality=0</code> and an <code>owl:maxCardinality=1</code> cardinality constraints in an object property $\text{Researcher} \equiv (\leq 1 \text{ hasAffiliation} \sqcap \geq 0 \text{ hasAffiliation})$
<b>ISB08</b>	Import a class which is equivalent to an anonymous class defined by an <code>owl:minCardinality=0</code> cardinality constraint in a datatype property $\text{Person} \equiv \geq 0 \text{ hasName}$
<b>ISB09</b>	Import a class which is equivalent to an anonymous class defined by an <code>owl:maxCardinality=1</code> cardinality constraint in a datatype property $\text{Researcher} \equiv \leq 1 \text{ wrotePhDThesis}$
<b>ISB10</b>	Import a class which is equivalent to an anonymous class defined by an <code>owl:cardinality=1</code> cardinality constraint in a datatype property $\text{Person} \equiv 1 \text{ hasSSN}$
<b>ISB11</b>	Import a class which is equivalent to an anonymous class defined by an <code>owl:minCardinality=0</code> and an <code>owl:maxCardinality=1</code> cardinality constraints in a datatype property $\begin{aligned} \text{Researcher} &\equiv \geq 0 \text{ wrotePhDThesis} \\ \text{Researcher} &\equiv \leq 1 \text{ wrotePhDThesis} \end{aligned}$
<b>ISB12</b>	Import a class which is equivalent to an anonymous class defined by the intersection of two other classes $\text{ItalianMan} \equiv (\text{Italian} \sqcap \text{Male})$

**Group C: Class defined by set operators**

<b>ISC01</b>	Import a class which is intersection of two other classes $\text{ItalianMan} \equiv (\text{Italian} \sqcap \text{Male})$
<b>ISC02</b>	Import a class which is intersection of several other classes $\text{ItalianMan} \equiv (\text{Italian} \sqcap \text{Male} \sqcap \text{Person})$

**Property benchmarks****Group D: Property hierarchies**

<b>ISD01</b>	Import a single object property <i>hasChild</i>
<b>ISD02</b>	Import an object property that is subproperty of another object property that is subproperty of a third one <i>isFatherOf</i> $\sqsubseteq$ <i>isGrandFatherOf</i> $\sqsubseteq$ <i>isAncestorOf</i>
<b>ISD03</b>	Import a single datatype property <i>hasAge</i>
<b>ISD04</b>	Import a datatype property that is subproperty of another datatype property that is subproperty of a third one <i>isInteger</i> $\sqsubseteq$ <i>isRational</i> $\sqsubseteq$ <i>isReal</i>

### Group E: Properties with domain and range

<b>ISE01</b>	Import a single object property with domain a class $\top \sqsubseteq \forall hasChild^-.Person$
<b>ISE02</b>	Import a single object property with range a class $\top \sqsubseteq \forall hasChild.Person$
<b>ISE03</b>	Import a single object property with domain a class and range another class $\top \sqsubseteq \forall hasChild^-.Father$ $\top \sqsubseteq \forall hasChild.Person$
<b>ISE04</b>	Import a single object property with domain and range the same class $\top \sqsubseteq \forall hasChild^-.Person$ $\top \sqsubseteq \forall hasChild.Person$
<b>ISE05</b>	Import a single object property with domain multiple classes and range a class $\top \sqsubseteq \forall hasChild^-.Mother$ $\top \sqsubseteq \forall hasChild^-.Woman$ $\top \sqsubseteq \forall hasChild^-.Person$ $\top \sqsubseteq \forall hasChild.Person$
<b>ISE06</b>	Import a single object property with domain a class and range multiple classes $\top \sqsubseteq \forall hasChild^-.Person$ $\top \sqsubseteq \forall hasChild.Person$ $\top \sqsubseteq \forall hasChild.Human$ $\top \sqsubseteq \forall hasChild.Child$
<b>ISE07</b>	Import a single datatype property with domain a class $\top \sqsubseteq \forall hasSSN^-.Person$

(continued on next page)

(continued from previous page)

<b>ISE08</b>	Import a single datatype property with range <code>rdfs:Literal</code> $\top \sqsubseteq \forall \text{hasName}.\text{rdfs:Literal}$
<b>ISE09</b>	Import a single datatype property with domain a class and range <code>rdfs:Literal</code> $\top \sqsubseteq \forall \text{hasName}^-. \text{Person}$ $\top \sqsubseteq \forall \text{hasName}.\text{rdfs:Literal}$
<b>ISE10</b>	Import a single datatype property with domain multiple classes and range <code>rdfs:Literal</code> $\top \sqsubseteq \forall \text{hasChildNamed}^-. \text{Mother}$ $\top \sqsubseteq \forall \text{hasChildNamed}^-. \text{Woman}$ $\top \sqsubseteq \forall \text{hasChildNamed}.\text{rdfs:Literal}$

**Group F: Property equivalences**

<b>ISF01</b>	Import several object properties with domain a class and range another class, which are all of them equivalent $\top \sqsubseteq \forall \text{livesIn}^-. \text{Person}$ $\top \sqsubseteq \forall \text{livesIn}^-. \text{City}$ $\text{livesIn} \equiv \text{isResdentIn}$
<b>ISF02</b>	Import several datatype properties with domain a class and range <code>rdfs:Literal</code> , which are all of them equivalent $\top \sqsubseteq \forall \text{hasName}^-. \text{City}$ $\top \sqsubseteq \forall \text{hasName}.\text{rdfs:Literal}$ $\text{hasName} \equiv \text{hasSpanishName}$
<b>ISF03</b>	Import an object property with domain a class and range another class, which is inverse of another object property $\top \sqsubseteq \forall \text{hasParent}^-. \text{Child}$ $\top \sqsubseteq \forall \text{hasParent}^-. \text{Person}$ $\text{hasChild} \equiv \text{hasParent}^-$

**Group G: Logical characteristics of properties**

<b>ISG01</b>	Import a single transitive object property with domain and range the same class $\text{hasFriend}^+ \sqsubseteq \text{hasFriend}$ $\top \sqsubseteq \forall \text{hasFriend}^-. \text{Person}$ $\top \sqsubseteq \forall \text{hasFriend}.\text{Person}$
--------------	---

(continued on next page)

*(continued from previous page)*

<b>ISG02</b>	Import a single symmetric object property with domain and range the same class $hasFriend \equiv hasFriend^-$ $\top \sqsubseteq \forall hasFriend^-.Person$ $\top \sqsubseteq \forall hasFriend.Person$
<b>ISG03</b>	Import a single functional object property with domain a class and range another class $\top \sqsubseteq \forall hasHusband$ $\top \sqsubseteq \forall hasHusband^-.Woman$ $\top \sqsubseteq \forall hasHusband.Man$
<b>ISG04</b>	Import a single functional datatype property with domain a class and range <code>rdfs:Literal</code> $\top \sqsubseteq \forall hasAge$ $\top \sqsubseteq \forall hasAge^-.Person$ $\top \sqsubseteq \forall hasAge.rdfs:Literal$
<b>ISG05</b>	Import a single inverse functional object property with domain a class and range another class $\top \sqsubseteq \forall hasTutor^-$ $\top \sqsubseteq \forall hasTutor^-.Professor$ $\top \sqsubseteq \forall hasTutor.Student$

## Individual benchmarks

### Group H: Single individuals

<b>ISH01</b>	Import one class and one individual that is instance of the class Person(PETER)
<b>ISH02</b>	Import several classes and one individual that is instance of all of them Person(PETER) Father(PETER) Student(PETER)
<b>ISH03</b>	Import one class and several individuals that are instance of the class Person(PETER) Person(PAUL) Person(MARY)

### Group I: Named individuals and properties



<b>ISI01</b>	<p>Import one class, one object property with domain and range the class, and one individual of the class that has the object property with another individual of the same class</p> <div> <math>\top \sqsubseteq \forall hasChild^-.Person</math>  <math>\top \sqsubseteq \forall hasChild.Person</math>  <math>Person(MARY)</math>  <math>Person(PAUL)</math>  <math>hasChild(MARY, PAUL)</math> </div>
<b>ISI02</b>	<p>Import one class, one object property with domain and range the class, and one individual of the class that has the object property with himself</p> <div> <math>\top \sqsubseteq \forall hasChild^-.Person</math>  <math>\top \sqsubseteq \forall hasChild.Person</math>  <math>Person(PAUL)</math>  <math>knows(PAUL, PAUL)</math> </div>
<b>ISI03</b>	<p>Import two classes, one object property with domain one class and range the other class, and one individual of one class that has the object property with an individual of the other class</p> <div> <math>\top \sqsubseteq \forall hasChild^-.Mother</math>  <math>\top \sqsubseteq \forall hasChild.Child</math>  <math>Mother(MARY)</math>  <math>Child(PAUL)</math>  <math>hasChild(MARY, PAUL)</math> </div>
<b>ISI04</b>	<p>Import one class, one datatype property with domain the class and range <code>rdfs:Literal</code>, and one individual of the class that has the datatype property with a literal</p> <div> <math>\top \sqsubseteq \forall hasName^-.Person</math>  <math>\top \sqsubseteq \forall hasName.rdfs:Literal</math>  <math>Person(MARYSMITH)</math>  <math>hasName(MARYSMITH, "Mary")</math> </div>
<b>ISI05</b>	<p>Import one class, one datatype property with domain the class and range <code>rdfs:Literal</code>, and one individual of the class that has the datatype property with several literals</p> <div> <math>\top \sqsubseteq \forall hasName^-.Person</math>  <math>\top \sqsubseteq \forall hasName.rdfs:Literal</math>  <math>Person(MARYANN)</math>  <math>hasName(MARYANN, "Mary")</math>  <math>hasName(MARYANN, "Ann")</math> </div>

### Group J: Anonymous individuals and properties

<b>ISJ01</b>	<p>Import one class, one object property with domain and range the class, and one anonymous individual of the class that has the object property with another individual of the same class</p> <div> <math display="block">\top \sqsubseteq \forall hasChild^-.Person</math> <math display="block">\top \sqsubseteq \forall hasChild.Person</math> <math display="block">Person(JOHN)</math> <math display="block">hasChild(ANON^a, JOHN)</math> </div> <hr/> <p><sup>a</sup>This denotes an <i>anonymous individual</i></p>
<b>ISJ02</b>	<p>Import two classes, one object property with domain one class and range the other class, and one anonymous individual of one class that has the datatype property with an individual of the other class</p> <div> <math display="block">\top \sqsubseteq \forall hasChild^-.Parent</math> <math display="block">\top \sqsubseteq \forall hasChild.Person</math> <math display="block">Person(JOHN)</math> <math display="block">hasChild(ANON, JOHN)</math> </div>
<b>ISJ03</b>	<p>Import one class, one datatype property with domain the class and range <code>rdfs:Literal</code>, and one anonymous individual of the class that has the datatype property with a literal</p> <div> <math display="block">\top \sqsubseteq \forall hasName^-.Person</math> <math display="block">\top \sqsubseteq \forall hasName.rdfs:Literal</math> <math display="block">hasName(ANON, "Peter")</math> </div>

### Group K: Individual identity

<b>ISK01</b>	<p>Import one class and two named individuals of the class that are the same</p> <div> <math display="block">Person(MARYANN) = Person(MARY)</math> </div>
<b>ISK02</b>	<p>Import one class and two named individuals of the class that are different</p> <div> <math display="block">\neg (Person(MARYANN) = Person(MARY))</math> </div>
<b>ISK03</b>	<p>Import one class and three named individuals of the class that are all of them different</p> <div> <math display="block">\neg (Person(MARY) = Person(ANN))</math> <math display="block">\neg (Person(MARY) = Person(JOAN))</math> <math display="block">\neg (Person(JOAN) = Person(ANN))</math> </div>

# Appendix C

## The *benchmarkOntology* and *resultOntology* ontologies

### The *benchmarkOntology* ontology

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:bo="http://knowledgeweb.semanticweb.org/owl/benchmarkOntology#"
  xml:base="http://knowledgeweb.semanticweb.org/owl/benchmarkOntology#">

  <owl:Ontology rdf:about="http://knowledgeweb.semanticweb.org/owl/benchmarkOntology#">
    <rdfs:comment>This ontology contains a description
      of the benchmark suite inputs.</rdfs:comment>
    <owl:versionInfo>24 October 2006</owl:versionInfo>
  </owl:Ontology>

  <!-- classes -->

  <owl:Class rdf:about="#Benchmark">
  </owl:Class>

  <owl:Class rdf:about="#Document">
  </owl:Class>

  <!-- properties -->

  <owl:ObjectProperty rdf:about="#usesDocument">
    <rdfs:domain rdf:resource="#Benchmark"/>
    <rdfs:range rdf:resource="#Document"/>
  </owl:ObjectProperty>

  <owl:DatatypeProperty rdf:about="#interchangeLanguage">
    <rdfs:domain rdf:resource="#Benchmark"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#id">
    <rdfs:domain rdf:resource="#Benchmark"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
```

```

</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#author">
  <rdfs:domain rdf:resource="#Benchmark"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#version">
  <rdfs:domain rdf:resource="#Benchmark"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#documentURL">
  <rdfs:domain rdf:resource="#Document"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#ontologyName">
  <rdfs:domain rdf:resource="#Document"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#ontologyNamespace">
  <rdfs:domain rdf:resource="#Document"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#representationLanguage">
  <rdfs:domain rdf:resource="#Document"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
</rdf:RDF>

```

## The *resultOntology* ontology

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:ro="http://knowledgeweb.semanticweb.org/owl/resultOntology#"
  xml:base="http://knowledgeweb.semanticweb.org/owl/resultOntology#">

  <owl:Ontology rdf:about="http://knowledgeweb.semanticweb.org/owl/resultOntology#">
    <rdfs:comment>This ontology contains a description of
    the benchmark suite results.</rdfs:comment>
    <owl:versionInfo>24 October 2006</owl:versionInfo>
  </owl:Ontology>

  <!-- classes -->

  <owl:Class rdf:about="#Tool">
  </owl:Class>

  <owl:Class rdf:about="#BenchmarkExecution">
  </owl:Class>

  <owl:Class rdf:about="#Result">
  </owl:Class>

  <!-- subclasses -->

```

```
<owl:Class rdf:about="#Step1Result">
  <rdfs:subClassOf rdf:resource="#Result" />
</owl:Class>

<owl:Class rdf:about="#Step2Result">
  <rdfs:subClassOf rdf:resource="#Result" />
</owl:Class>

<owl:Class rdf:about="#FinalResult">
  <rdfs:subClassOf rdf:resource="#Result" />
</owl:Class>

<!-- properties -->

<owl:ObjectProperty rdf:about="#hasStep1Result">
  <rdfs:domain rdf:resource="#BenchmarkExecution" />
  <rdfs:range rdf:resource="#Result" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasStep2Result">
  <rdfs:domain rdf:resource="#BenchmarkExecution" />
  <rdfs:range rdf:resource="#Result" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasFinalResult">
  <rdfs:domain rdf:resource="#BenchmarkExecution" />
  <rdfs:range rdf:resource="#Result" />
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:about="#toolName">
  <rdfs:domain rdf:resource="#Tool" />
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#toolVersion">
  <rdfs:domain rdf:resource="#Tool" />
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:about="#originTool">
  <rdfs:domain rdf:resource="#BenchmarkExecution" />
  <rdfs:range rdf:resource="#Tool" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#destinationTool">
  <rdfs:domain rdf:resource="#BenchmarkExecution" />
  <rdfs:range rdf:resource="#Tool" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#ofBenchmark">
  <rdfs:domain rdf:resource="#BenchmarkExecution" />
  <rdfs:range rdf:resource="#Benchmark" />
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:about="#interchange">
  <rdfs:domain rdf:resource="#Result" />
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#informationAdded">
  <rdfs:domain rdf:resource="#Result" />
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
</owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:about="#informationRemoved">
  <rdfs:domain rdf:resource="#Result"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#execution">
  <rdfs:domain rdf:resource="#Result"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#timestamp">
  <rdfs:domain rdf:resource="#BenchmarkExecution"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#datetime"/>
</owl:DatatypeProperty>

</rdf:RDF>
```

# 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.
- [Bull *et al.*, 1999] J. M. Bull, L. A. Smith, M. D. Westhead, D. S. Henty, and R. A. Davey. A methodology for benchmarking java grande applications. In *Proceedings of the ACM 1999 conference on Java Grande*, pages 81–88, 1999.
- [David *et al.*, 2006] S. David, R. García-Castro, and A. Gómez-Pérez. Defining a benchmark suite for evaluating the import of OWL lite ontologies. In *Proceedings of the OWL: Experiences and Directions 2006 workshop (OWL2006)*, Athens, Georgia, USA, November 10-11 2006.
- [García-Castro and Gómez-Pérez, 2005] 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 *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 *et al.*, 2006] R. García-Castro, Y. Sure, M. Zondler, O. Corby, J. Prieto-González, E. Paslaru Bontas, L. Nixon, and M. Mochol. D1.2.2.1.1 benchmarking the interoperability of ontology development tools using rdf(s) as interchange language. Technical report, Knowledge Web, June 2006.
- [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.
- [García-Castro, 2007] R. García-Castro. D6.8.1 testing the neon toolkit interoperability. Technical report, NeOn, September 2007.

- [Guo *et al.*, 2005] Y. Guo, Z. Pan, and J. Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. *Journal of Web Semantics* 3(2), (2):158–182, 2005.
- [Ma *et al.*, 2006] Li Ma, Yang Yang, Zhaoming Qiu, GuoTong Xie, Yue Pan, and Shengping Liu. Towards a complete OWL ontology benchmark. In Y. Sure and J. Domingue, editors, *Proceedings of the 3rd European Semantic Web Conference (ESWC 2006)*, volume 4011 of *LNCS*, pages 125–139, Budva, Montenegro, June 11-14 2006.
- [McGuinness and van Harmelen, 2004] D.L. McGuinness and F. van Harmelen. OWL web ontology language overview. Technical report, W3C, 10 February 2004.
- [Shirazi *et al.*, 1999] B. Shirazi, L.R. Welch, B. Ravindran, C. Cavanaugh, B. Yanamula, R. Brucks, and E. Huh. Dynbench: A dynamic benchmark suite for distributed real-time systems. In *Proc. of the 11 IPPS/SPDP'99 Workshops*, pages 1335–1349. Springer-Verlag, 1999.
- [Sim *et al.*, 2003] S. Sim, S. Easterbrook, and R. Holt. Using benchmarking to advance research: A challenge to software engineering. In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*, pages 74–83, Portland, OR, 2003.
- [Stefani *et al.*, 2003] F. Stefani, D. Macii, A. Moschitta, and D. Petri. FFT Benchmarking for Digital Signal Processing Technologies. In *17th IMEKO World Congress*, Dubrovnik, Croatia, 22-27 June 2003.
- [Volz, 2004] Rapahel Volz. *Web ontology reasoning with logic databases*. PhD thesis, AIFB Karlsruhe, 2004.



# Acknowledgments

Thanks to all the people that have participated in the OWL interoperability benchmarking by adapting the IBSE tool for some best-in-class Semantic Web tools: Stamatia Dasiopoulou, Danica Damjanovic, Michael Erdmann, Christian Fillies, Roman Korf, Diana Maynard, York Sure, Jan Wielemaker, and Philipp Zaltenbach. 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	D2.1.4	<b>Specification of a methodology, general criteria and benchmark suites for benchmarking ontology tools</b> presented the benchmarking methodology that has been used for benchmarking the interoperability of ontology development tools using RDF(S) and OWL as interchange languages.
KW	D1.2.2.1.1	<b>Benchmarking the interoperability of ontology development tools using RDF(S) as interchange language</b> described the benchmarking of the interoperability of ontology development tools using RDF(S) as interchange language that took place in Knowledge Web, including the analysis of the results obtained.